RAMS
The Regional Atmospheric Modeling System

Version 3b

User's Guide

RAMS
The Regional Atmospheric Modeling System

Version 3b

User's Guide

by

Robert L. Walko, Craig J. Tremback, and Rolf F.A. Hertenstein

***ASTER Division**
Mission Research Corporation
P.O. Box 466
Fort Collins, CO 80522

October, 1995

RAMS User Guide

Contents

CHAPTER 1 - INTRODUCTION

RAMS, the Regional Atmospheric Modeling System, is a highly versatile numerical code developed by scientists at Colorado State University and the *ASTER division of Mission Research Corporation for simulating and forecasting meteorological phenomena, and for depicting the results. Its major components are (1) an atmospheric model which performs the actual simulations, (2) a data analysis package which prepares initial data for the atmospheric model from observed meteorological data, and (3) a post-processing model visualization and analysis package which interfaces atmospheric model output with a variety of visualization software utilities. The atmospheric model is constructed around the full set of primitive dynamical equations which govern atmospheric motions, and supplements these equations with optional parameterizations for turbulent diffusion, solar and terrestrial radiation, moist processes including the formation and interaction of clouds and precipitating liquid and ice hydrometeors, sensible and latent heat exchange between the atmosphere, multiple soil layers, a vegetation canopy, and surface water, the kinematic effects of terrain, and cumulus convection. RAMS is fundamentally a limited-area model, but may be configured to cover an area as large as a planetary hemisphere for simulating mesoscale and large scale atmospheric systems. There is no lower limit to the domain size or to the mesh cell size of the model's finite difference grid; microscale phenomena such as tornadoes and boundary layer eddies, as well as submicroscale turbulent flow over buildings and in a wind tunnel, have been simulated with this code. Two-way interactive grid nesting in RAMS allows local fine mesh grids to resolve compact atmospheric systems such as thunderstorms, while simultaneously modeling the large scale environment of the systems on a coarser grid. RAMS is now primarily supported for execution under the UNIX operating system and its variants such as UNICOS. Formerly it also ran under COS, VMS, and CTSS but there has been little demand for these in recent years. The code is written almost exclusively in FORTRAN 77 with some common extensions, although some C code is used to facilitate I/O procedures and dynamic memory allocation. In order to utilize the standard graphics capability of RAMS, the computer installation should have (at least) the GKS Version 3.0 of NCAR Graphics.

RAMS is an outgrowth of two earlier atmospheric modeling programs conducted independently during the 1970's. A cloud model developed under the direction of Dr. William R. Cotton contributed state-of-the-art methods for modeling microscale dynamic systems and microphysical processes. A mesoscale model developed under the direction of Dr. Roger A. Pielke contributed expertise in the modeling of mesoscale systems and the influence of land surface characteristics on the atmosphere. In 1986, the process was begun of merging the capabilities of the two models into a unified multi-purpose modeling system, and thus was born the new RAMS code. In order to introduce a high degree of flexibility and versatility in RAMS, particularly regarding its new grid nesting capability, and to take advantage of the ever-increasing capabilities in computer hardware and software, RAMS was built on an entirely new framework, with the numerical schemes and parameterizations from the earlier models adapted to the new model structure. After two years of concerted effort, the first Version of the new RAMS code was in use as a research tool. A major program of continued development has continued to the present day, resulting in many improvements and new capabilities. The planning, design, and construction of the RAMS code has been conducted primarily by Drs. Craig J. Tremback and Robert L. Walko. This effort has been carried out with a major emphasis given to uniformity of design of the code, and nearly all developments have involved cross-discussion and/or debate which we hope has resulted in the best of our ideas being incorporated. Many valuable ideas and experiences with RAMS have been shared by the students of Drs. Pielke and Cotton and by other users over the years, which has led to significant improvements in RAMS.

This document is the third edition of the RAMS User's Guide, and describes Version 3b of RAMS which was released in August 1995. The first edition of the User's Guide was written for the release of Version 2c of RAMS in April 1991. Code updates to Version 2c were distributed in March 1992 with a brief set of Update Notes. Version 3a of RAMS, and the second version of the User's Guide were completed in December, 1993. The User's Guide is intended as an aid to those who need to install RAMS on a computer system and/or run the model as a tool for studying atmospheric processes. It does not delve into many of the more technical aspects of the code.

While versions of RAMS have been configured to run efficiently as *operational* models, this master version is used most often as a *research* model. The distinction implies that a very wide range of options is available for the user to select in configuring a model simulation or forecast, many or most of which are inappropriate for a particular application. The main reasons for the availability of options fall into two categories. First, certain options are required for some applications but not for others. For example, the parameterization of cumulus convection is intended for use on coarse grids where convective currents are not resolvable, whereas on fine grids, the model can simulate convection explicitly and cumulus convection should not be parameterized. Second, a variety of options is often made available to allow experimentation and testing of different parameterization schemes or parameter settings, as part of the ongoing research of improving atmospheric models. A prime example here is a set of new parameters to control four-dimensional data assimilation into the model. The traditional operational model is, by contrast, a black-box code in which most details of the model configuration are not accessible to the operator and the model is configured for a specific application. To the user, this means that the RAMS code can be misused. The research version of RAMS should not be treated as a black box; as a minimum, it should be used as a "gray box", requiring considerable knowledge by the user for proper setting of the model flags and parameters. It is the primary intent of this documentation to guide the user in setting up and running a model simulation. We have thus frequently injected our own experiences and advice, particularly in describing how to set values of the atmospheric model namelist variables in Chapter 7, but it should be remembered that it is far from possible to cover all avenues in such descriptions. Our experience has been that additional direct consultation is usually required for a user to come up to speed in using RAMS.

The remainder of this document is organized as follows: Chapter 2 lists the major components of RAMS, describing their basic functions and relation to each other. It also outlines the organization of RAMS into subdirectories and files, describing the function of each file with emphasis on those the user must become most familiar with. Chapter 3 provides instructions on installing RAMS on a computer running a UNIX operating system. Chapter 4 details the UNIX Makefiles and executable files in RAMS, which the user submits for optionally modifying standard RAMS code and executing a RAMS simulation. It also carries the user through a step-by-step description of how to input and objectively analyze observational data to initialize a RAMS simulation, set up and run the simulation, and produce graphical plots of simulation results. Chapter 5 describes the various datasets, mandatory and optional, which are input to or output from the major components of RAMS. Chapter 6 discusses how to set each of the namelist variables which control the operation of the isentropic analysis package (ISAN) and the DATAPREP package that prepares data for ISAN. Chapter 7 discusses how to set each of the namelist variables which control the operation of the atmospheric model. Chapter 8 discusses how to set each of the namelist variables which control the operation of the post-processing analysis packages (REVU and VAN). Finally, Chapter 9 discusses how to make the most commonly required modifications to the atmospheric model code to customize a simulation beyond what is capable through the namelist variables.

Section 1.1 - Changes For RAMS Version 3b

This section describes the major changes made to RAMS in its evolution from Version 3a to Version 3b.

- The new RAMS microphysics package, which has been used experimentally at Colorado State University since late 1993, has been included as the standard parameterization in version 3b. The older microphysics package is still an option, although use of the newer package is now strongly encouraged and support of the older package will be discontinued in the future. The major features of the new package are described in Chapter 7 as part of the individual descriptions of the namelist parameters that control the microphysics. Those parameters are ICLOUD, IRAIN, IPRIS, ISNOW, IAGGR, IGRAUP, IHAIL, CPARM, RPARM, PPARM, SPARM, APARM, GPARM, HPARM, INUCPRG, GHU, AMI0, COLTABFN, EVPTABFN, and MKCOLTAB.
- The ISAN package has been modified so that it requires upper air sounding and surface input data to be read from files written in a RAMS-specific format. The subroutines formerly

contained in ISAN that read data in the format obtained from the National Center for Atmospheric Research (NCAR), plus the routines formerly in ISAN intended for user modification to directly input data of various formats, have now been broken out into stand-alone programs. Thus, to get sounding and surface data into RAMS, one must first convert it to the RAMS-specific format using one of the stand-alone programs, and write it to files of the type recognized by ISAN. Then, ISAN is executed, which reads those files and generates initial fields for the model. This method of standardization greatly facilitates the process of identifying any problems with input data, since the data must first be put into the standard-format files. Descriptions of the stand-alone programs that produce the standard-format files have been added to Chapter 6.

- Adding scalar variables to RAMS, which is done for representing special atmospheric constituents such as trace gases or aerosols, can now be done automatically in unlimited number. Formerly, up to 5 variables could be added automatically, while any additional variables required code changes. The section in Chapter 9 that described those code changes has been removed, while the description of namelist variable NADDSC in chapter 7 mentions the unlimited number.

- Some changes to the directory structure and files have been made in the standard installation of RAMS. All include files formerly located under the separate isan, model, and van subdirectories are now grouped together in a single subdirectory. A new binary (bin) subdirectory has been created which contains the model archive and executable files. These enable users to avoid having to recompile the code for most applications. Even if a user needs to modify some code, recompilation is required only for the file(s) containing the modifications, since the standard archive is available in the standard model directories for the remainder of the code. The new directory structure and procedures for compiling the entire code or only part of the code are detailed in Chapters 2, 3, and 4.

- A replacement for the VAN package, called REVU (for RAMS Evaluation and Visualization Utility) has been developed which has added features including color-filled maps and contoured fields and the ability to overlay more fields. In version 3b, both VAN and REVU are available.

- Both VAN and REVU have an option to plot fields on standard constant pressure surfaces (1000, 900, 850, 700, 600, 500, 400, 300, 200, and 100 mb).

- Horizontal movement of nested grids is now operational.

- Model version identifiers (e.g. 3b) have been dropped from source code filenames which facilitates, among other things, future updates to this document.

- The module ruser.f, which is where various surface characteristics are optionally defined, has been modified both in content and purpose. Now, all surface characteristics, which include (1) topography, (2) land percentage, (3) vegetation type, (4) soil type, (5) soil roughness length, (6) sea surface temperature, (7) initial soil temperature, (8) initial soil moisture, (9) initial vegetation temperature, (10) initial vegetation surface moisture, and (11) a new simple snowcover parameter, are always initialized in rsurf.f for all grids. The initializations (or offsets) are horizontally homogeneous by default. A user may still modify these defaults to make inhomogeneous initial fields (e.g., by reading from files, setting individual points, etc.) in ruser.f, but by default the code in ruser.f is commented out and does nothing. Formerly that code set the default horizontally-homogeneous field values now done in rsurf.f. In addition, the four surface fields: topography, land percentage, sea surface temperature, and vegetation type may still be initialized from standard files by specifying appropriate namelist parameters. Ruser.f now also contains two template subroutines: bubble and trtend, the first of which can be used to customize an initial prognostic field, and the second of which can be used to add a forcing term to any prognostic field. These subroutines are commented out by default, and are described in Sections 9.2 and 9.3.

- Soil moisture is no longer initialized in rvari.f, but is initialized in ruser.f as described above. The namelist soil moisture will now have an effect in variable initialization runs.

- The Chen and Cotton longwave radiation scheme has been sped up considerably. This was done by removing the ability to treat fractional cloud cover (within any single grid cell), which was never used in practice but consumed much computation time because it involved an inner loop.
- The Chen and Cotton shortwave radiation scheme has been modified in several ways, resulting in improved fluxes.
- The turbulence option where namelist parameter idiffk is set to 3 has been modified slightly so that the horizontal and vertical diffusion coefficients (which are identical in this option) use only vertical diffusion parameters defined in the namelist, rather than a combination of vertical and horizontal diffusion parameters.
- Options now exist to output RAMS history files in either binary or ASCII format. This is further explained in the description of namelist variable IOUTPUT in Chapter 7. Version 3a had only the ASCII format as an option.
- We are now fully supporting use of Makefiles for running RAMS, VAN, and REVU, and have discontinued use of the script jobfile. Chapters 3 and 4 describe the Makefiles and their use in detail.
- A simple snowcover array has been implemented temporarily while the detailed RAMS snow model undergoes further testing and development. The simple snowcover is not prognosed or updated in RAMS; it is intended to be initialized from observational data, and serves to modify the surface albedo and thermal conductivity between the ground and atmosphere.
- In horizontally homogeneous initialization, the lowest sounding level was formerly assumed to apply at sea level. Now, a new namelist variable, HS, is included in which to specify the elevation of the lowest sounding level. This level must be lower or equal to the lowest terrain height in the model, so a user must be careful to choose a sounding appropriately for a given domain, or fabricate the necessary lower elevation sounding data.
- To help avoid confusion, the former RAMSIN namelist file used as input to the van package is now called VAN_IN, while the namelist file for the newer REVU package is REVU_IN.
- The script file lib.job has been removed. Now, the Makefile used to compile RAMS accesses the library files directly and includes them in the standard archive.

Section 1.2 - Ongoing Development

There are several ongoing development efforts at Colorado State University and the *ASTER division of Mission Research Corporation which will bring added features and improvements to RAMS over the coming months. Some of the major efforts currently in progress include:

- **Parallelization.** The RAMS code structure is being changed to allow for the parallel computation on a MIMD computer architecture, of which a workstation cluster is one example. Initial tests with a prototype version are very promising.
- **Microphysics.** While a major new microphysics package has been incorporated as part of version 3b, further major improvements of the new code are underway. One is that explicit prediction, rather than diagnosis, of cloud water mixing ratio is being developed, along with explicit prediction of multiple distributions of cloud condensation nuclei. A major application of this work is to predict atmospheric visibility, particularly as influenced by haze. The other major development is explicit prediction of number concentration for all hydrometeor categories, which will be added to the current prediction of all hydrometeor mass mixing ratios.

- **Radiation.** A new longwave and shortwave radiation package is being developed which should increase the accuracy and efficiency of the radiation parameterizations.
- **Cumulus Parameterization.** The scheme developed by Weissbluth is being standardized for Version 3 of RAMS.
- **Vegetation Parameterization.** The current vegetation parameterization is being merged with LEAF (developed by John Lee) with new features such as prognostic snow cover and multiple vegetation classes in a grid cell.
- **Hydrology.** The RAMS soil model is being coupled with TOPMODEL, a hydrology model that represents surface and subsurface lateral (downslope) flux of water.

In addition to developments to the basic RAMS code, other software products will be available including:

- **RINGI -** RAMS Interactive NCAR Graphics Interface will provide an X-window interface into the plotting capabilities of NCAR Graphics for the RAMS output files.
- **HYPACT -** HYbrid Particle And Concentration Transport package is *ASTER's new LaGrangian particle model with the ability to also transport tracer concentrations in Eulerian space.

CHAPTER 2 - PRINCIPAL COMPONENTS OF RAMS AND FILE ORGANIZATION

This chapter gives an overview of the major components of RAMS in terms of their functionality, the source code files that comprise each part, and the directory structure in which the files are organized.

Section 2.1 - Functional Components of RAMS

There are four main functional parts of RAMS:

1. The **model** is the prognostic component of RAMS. It advances gridded fields of atmospheric velocity, pressure, potential temperature, and optionally, moisture and other variables from an initial state through a series of discrete time steps to a future state based on a closed set of differential equations which describe the physical laws governing the atmosphere. A prognostic soil and vegetation model may optionally interact with the modeled atmosphere as a lower boundary condition. The model domain may be either two- or three-dimensional, and a small two-dimensional domain may be set up to function as a one-dimensional (vertical only) model.

2. The **ISentropic ANalysis package**, or **ISAN**, ingests observational atmospheric data of various types including rawinsondes, surface observations, and gridded pressure level analyses, objectively analyzes these data, and generates from them gridded three-dimensional fields of wind velocity, potential temperature, and water vapor mixing ratio. The resultant fields are used by the model as initial fields and, optionally, as time-dependent fields toward which the model solution is nudged in a 4-dimensional data assimilation scheme during time integration. Output from ISAN is required when the atmospheric model is run with the namelist variable INITIAL set to 2 (see description in Chapter 7), but not otherwise.

3. The **DATAPREP package** reads observational data in whatever format they are in and outputs them to files of a specific format for ingestion by ISAN. One version of DATAPREP specifically reads data in the form archived at the National Center for Atmospheric Research (NCAR), while another is a template to be modified for any generic data type.

4. The **postprocessing routines**, consisting of the RAMS Evaluation and Visualization Utility **REVU** and the Visualization and ANalysis package **VAN**, accept gridded fields output from the model and generate various types of graphical plots of the fields using NCAR graphics. REVU is the new replacement for VAN, but both are included in this RAMS version. REVU normally executes faster than VAN, but is still under development. It is now capable of producing 2-D contour and color-fill plots, vector or wind barb plots, and streamline plots of cross sections of model fields taken with respect to any two of the spatial coordinates (x,y,z).

Several minor components of RAMS include routines to generate standard input data files from arbitrarily-formatted data, to compile source code and build object file libraries, and some standard data files.

Section 2.2 - The Directory Structure and Files

The rams directory structure contains one main directory **rams3b**, 10 subdirectories immediately below it, **model, lib, isan, dprep, mrc, van, include, bin, util,** and **data**, and 9 subdirectories at the level below, **post** and **revu** under mrc, **generic** and **ncar** under dprep, **libc** under lib, and **micro, elev10m, pctl10m**, and **sst** under data. Following is a complete listing of all subdirectories and the files within them:

`rams3b/model/`

```
RAMSIN      raco.f      radvf.f     radvl.f     rammain.f   rbnd.f
rcio.f      rconv.f     rdriv.f     rhhi.f      rhyd.f      rinit.f
rio.f       rmcth.f     rmicr.f     rmisc.f     rname.f     rnest.f
rprnt.f     rsurf.f     rthrm.f     rtimh.f     rturb.f     ruser.f
rusgs.f     rvari.f     rvtab.f

rams3b/lib/
libc/       rfft.f      rfvec.f     rmcn.f      rmcro.f     rmdg.f
rmflw.f     rmutl.f     rname.f     rnumr.f     rpolr.f     rrad.f
rsys.F      rutil.f     rvfor.f

rams3b/lib/libc/
dirs.c        ramsutils.c   sub_names.h

rams3b/isan/
aobj.f      aplt.f      asgen.f     askewt.f    astga.f     astgb.f
asti.f      asti2.f     astp.f      avarf.f

rams3b/dprep/
generic/  ncar/

rams3b/dprep/generic/
Makefile*      PREP_IN         asgen.f         namelist.info
prep_com.h     press_read.f    read_rawi.f     read_sfc.f

rams3b/dprep/ncar/
Makefile*      PREP_IN      asgen.f      asti2.f      astp.f
prep_com.h     rassim.f

rams3b/mrc/
post/  revu/

rams3b/mrc/post/
RAMSfilelist.f   barbs.f          charutils.f      cread_RAMS.f
defparse.f       dumpout.f        eenviron.c       error_mess.f
hutil.f          hvlib.f          iplt.f           mkmap.f
plt.f            rainit.f         ramscode.f       ramsgks.f
rcolors.f        rvec.f

rams3b/mrc/revu/
REVU_IN*      rbat.f   revu_main.f   rgrab.f

rams3b/van/
RAMSIN       rafld.f      rcnfa.f      rplt.f       van_main.f

rams3b/include/
ascoms.h       rcommons.h     rconfig.h      rconstants.h   vcomm2.h
vcommons.h

rams3b/bin/
Makefile*    Make.rams*   Make.lib*    Make.revu*    Make.van*    RAMS_3b*
RAMS_3b.a    RAMS_van*    RAMS_van.a   Revu*         Revu.a       bat*
librams.a    make.err     make.lis

rams3b/util/
mktopo.f       ncar_dummy.f

rams3b/data/micro/
```

```
ct1.0   ct2.0   ct3.0   et1.0   et1.5   et2.0   et2.5   et3.0   et3.5   et4.0
et4.5   et5.0

rams3b/data/elev10m/
H10N000E   H10N020E   H10N020W   H10N040E   H10N040W   H10N060E   H10N060W
H10N080E   H10N080W   H10N100E   H10N100W   H10N120E   H10N120W   H10N140E
H10N140W   H10N160E   H10N160W   H10N180W   H10S000E   H10S020E   H10S020W
H10S040E   H10S040W   H10S060E   H10S060W   H10S080E   H10S080W   H10S100E
H10S100W   H10S120E   H10S120W   H10S140E   H10S140W   H10S160E   H10S160W
H10S180W   H30N000E   H30N020E   H30N020W   H30N040E   H30N040W   H30N060E
H30N060W   H30N080E   H30N080W   H30N100E   H30N100W   H30N120E   H30N120W
H30N140E   H30N140W   H30N160E   H30N160W   H30N180W   H30S000E   H30S020E
H30S020W   H30S040E   H30S040W   H30S060E   H30S060W   H30S080E   H30S080W
H30S100E   H30S100W   H30S120E   H30S120W   H30S140E   H30S140W   H30S160E
H30S160W   H30S180W   H50N000E   H50N020E   H50N020W   H50N040E   H50N040W
H50N060E   H50N060W   H50N080E   H50N080W   H50N100E   H50N100W   H50N120E
H50N120W   H50N140E   H50N140W   H50N160E   H50N160W   H50N180W   H50S000E
H50S020E   H50S020W   H50S040E   H50S040W   H50S060E   H50S060W   H50S080E
H50S080W   H50S100E   H50S100W   H50S120E   H50S120W   H50S140E   H50S140W
H50S160E   H50S160W   H50S180W   H70N000E   H70N020E   H70N020W   H70N040E
H70N040W   H70N060E   H70N060W   H70N080E   H70N080W   H70N100E   H70N100W
H70N120E   H70N120W   H70N140E   H70N140W   H70N160E   H70N160W   H70N180W
H70S000E   H70S020E   H70S020W   H70S040E   H70S040W   H70S060E   H70S060W
H70S080E   H70S080W   H70S100E   H70S100W   H70S120E   H70S120W   H70S140E
H70S140W   H70S160E   H70S160W   H70S180W   H90S000E   H90S020E   H90S020W
H90S040E   H90S040W   H90S060E   H90S060W   H90S080E   H90S080W   H90S100E
H90S100W   H90S120E   H90S120W   H90S140E   H90S140W   H90S160E   H90S160W
H90S180W   HHEADER

rams3b/data/pctl10m/
L10N000E   L10N020E   L10N020W   L10N040E   L10N040W   L10N060E   L10N060W
L10N080E   L10N080W   L10N100E   L10N100W   L10N120E   L10N120W   L10N140E
L10N140W   L10N160E   L10N160W   L10N180W   L10S000E   L10S020E   L10S020W
L10S040E   L10S040W   L10S060E   L10S060W   L10S080E   L10S080W   L10S100E
L10S100W   L10S120E   L10S120W   L10S140E   L10S140W   L10S160E   L10S160W
L10S180W   L30N000E   L30N020E   L30N020W   L30N040E   L30N040W   L30N060E
L30N060W   L30N080E   L30N080W   L30N100E   L30N100W   L30N120E   L30N120W
L30N140E   L30N140W   L30N160E   L30N160W   L30N180W   L30S000E   L30S020E
L30S020W   L30S040E   L30S040W   L30S060E   L30S060W   L30S080E   L30S080W
L30S100E   L30S100W   L30S120E   L30S120W   L30S140E   L30S140W   L30S160E
L30S160W   L30S180W   L50N000E   L50N020E   L50N020W   L50N040E   L50N040W
L50N060E   L50N060W   L50N080E   L50N080W   L50N100E   L50N100W   L50N120E
L50N120W   L50N140E   L50N140W   L50N160E   L50N160W   L50N180W   L50S000E
L50S020E   L50S020W   L50S040E   L50S040W   L50S060E   L50S060W   L50S080E
L50S080W   L50S100E   L50S100W   L50S120E   L50S120W   L50S140E   L50S140W
L50S160E   L50S160W   L50S180W   L70N000E   L70N020E   L70N020W   L70N040E
L70N040W   L70N060E   L70N060W   L70N080E   L70N080W   L70N100E   L70N100W
L70N120E   L70N120W   L70N140E   L70N140W   L70N160E   L70N160W   L70N180W
L70S000E   L70S020E   L70S020W   L70S040E   L70S040W   L70S060E   L70S060W
L70S080E   L70S080W   L70S100E   L70S100W   L70S120E   L70S120W   L70S140E
L70S140W   L70S160E   L70S160W   L70S180W   L90S000E   L90S020E   L90S020W
L90S040E   L90S040W   L90S060E   L90S060W   L90S080E   L90S080W   L90S100E
L90S100W   L90S120E   L90S120W   L90S140E   L90S140W   L90S160E   L90S160W
L90S180W   LHEADER

rams3b/data/sst/
SAPR90S000E   SAPR90S180W   SAPRHEADER    SAUG90S000E   SAUG90S180W
SAUGHEADER    SDEC90S000E   SDEC90S180W   SDECHEADER    SFEB90S000E
```

```
SFEB90S180W   SFEBHEADER    SHEADER       SJAN90S000E   SJAN90S180W
SJANHEADER    SJUL90S000E   SJUL90S180W   SJULHEADER    SJUN90S000E
SJUN90S180W   SJUNHEADER    SMAR90S000E   SMAR90S180W   SMARHEADER
SMAY90S000E   SMAY90S180W   SMAYHEADER    SNOV90S000E   SNOV90S180W
SNOVHEADER    SOCT90S000E   SOCT90S180W   SOCTHEADER    SSEP90S000E
SSEP90S180W   SSEPHEADER

rams3b/sample/model_output/
r.err              r.lis              s.a0s.g1           s.a0s.head
s.a14400s.g1       s.a14400s.head     s.a21600s.g1       s.a21600s.head
s.a28800s.g1       s.a28800s.head     s.a36000s.g1       s.a36000s.head
s.a43200s.g1       s.a43200s.head     s.a7200s.g1        s.a7200s.head

rams3b/sample/revu_output/
gmeta
```

The remaining Sections of Chapter 2 give brief descriptions of the individual files by directory.

Section 2.2.1 - the model subdirectory

The model subdirectory contains most of the source code files for the atmospheric model. Each file is a collection of subroutines which are usually grouped into the file according to the nature of their function.

RAMSIN is an input file read by the model and ISAN that contains namelist data. These data specify many aspects of the model configuration and options including, to name a few, dimensionality, number and sizes of grids, grid spacing, type of initialization, type of boundary conditions and parameterizations used, input and output filenames, whether isentropic objective analysis is to be done, and duration of the run. It also contains numerous data used in initializing atmospheric, soil, and vegetation fields and controlling physical parameters. Namelist variables specific to controlling ISAN are described in Chapter 6, and the others in Chapter 7.

raco.f contains the subroutines which perform the pressure gradient force computation and the pressure tendency computation based on the continuity equation. This set of computations is that which governs sound waves, and is performed on a small time-split timestep. The routines in **raco.f** are used only in conjunction with the nonhydrostatic option in the model.

radvf.f contains all subroutines involved in the advection computations used with the forward-in-time advection scheme.

radvl.f contains all subroutines involved in the advection computations used with the leapfrog schemes.

rammain.f consists only of the main program which calls the highest level subroutine in the model.

rbnd.f contains most of the model subroutines used in computing top, bottom, and lateral boundary conditions on prognostic variables.

rcio.f contains the read and write routine which performs I/O on selected common block information for the history and analysis files.

rconv.f contains all of the model subroutines involved in the convective parameterization in the model.

rdriv.f is the module containing the top level subroutines in the atmospheric model. Subroutine rams_driver is at the top, being called directly by the short main program. Subroutine rams_driver first calls several routines that first dynamically allocates needed memory, and then calls the two primary subroutines INITLZ and MODEL, both of which are also located in **rdriv.f**. Subroutine INITLZ drives the entire model initialization procedure. When INITLZ has been executed, all prognostic variables, grid indices and spacings, reference state variables, etc., have been initialized, and the model is ready for time integration. Subroutine MODEL is called next, and drives the entire time integration procedure. It contains the actual timestep loop. Inside this loop, a call is made to subroutine TIMESTEP which carries out the advancement of the model solution on one grid for one timestep. Also inside the timestep loop are subroutine calls to perform the necessary sequencing of grid nesting operations. Module **rdriv.f** also contains subroutines which make some checks on values assigned to namelist variables (see Chapter 7), and stop the model with the issue of an error message to the user if the variables checked are improperly set.

rhhi.f consists of subroutines uniquely involved in initializing a simulation whose initial atmospheric fields are interpolated horizontally homogeneously from a single sounding. This option is described more fully in the description of namelist variable INITIAL given in Section 7.3.

rhyd.f contains the subroutines that perform the pressure gradient force computation and the pressure tendency computation based on the continuity equation. This set of computations is that which governs Lamb waves, and is performed on a small time-split timestep. The routines in **rhyd.f** are used only in conjunction with the hydrostatic option in the model.

rinit.f contains several subroutines that perform various facets of initializing the model, or of initializing certain grid-dependent parameters whenever the model switches from one active grid to another.

rio.f contains subroutines that perform I/O operations in the model. These operations include the reading and writing of history and analysis files.

rmcth.f contains the higher-level and initialization subroutines for the new bulk microphysics parameterization.

rmicr.f contains the lower-level subroutines for the new bulk microphysics parameterization.

rmisc.f contains miscellaneous routines such as the radiation parameterization driver, and Coriolis force computations.

rname.f contains subroutines involved in reading the variables in the five atmospheric model namelists in the RAMSIN file (see Chapter 7).

rnest.f consists of subroutines that perform the two-way communication between different nested grids, including moving grids.

rprnt.f contains subroutines involved in computing various integral quantities of a model simulation, and writing those and other quantities to the standard output file generated from a model run.

rsurf.f contains all subroutines involved in the prognostic soil and vegetation parameterizations in the model, plus subroutines which perform initialization of surface, vegetation, and soil variables.

rthrm.f contains the higher-level subroutines that drive the old bulk microphysics parameterization.

rtimh.f contains subroutine TIMESTEP, whose function is to drive an entire single timestep on one grid. TIMESTEP calls the top-level drivers for the individual physical processes, such as advection and diffusion, accumulating the tendencies of prognostic variables computed in each process. Subroutine TIMESTEP is a good place for the user to add any required special forcing term to a prognostic equation (see Section 9.3).

rturb.f contains all subroutines involved in the various forms of turbulent diffusion parameterization available in RAMS.

ruser.f is a special module containing several subroutines where a user may customize initial prognostic fields or forcing terms for prognostic fields. These routines often require modification by the user and are thus grouped together (see Sections 9.1 - 9.3).

rusgs.f contains the code which processes the experimental USGS land use datasets.

rvari.f consists of subroutines uniquely involved in initializing a simulation whose initial atmospheric fields are interpolated horizontally inhomogeneously from 3-D observational atmospheric data. This option is described more fully in the description of namelist variable INITIAL given in Section 7.3.

rvtab.f consists of a single subroutine having the name VTABLES. This is a very important subroutine in the model; its function is to define the variables which will be used in the run by filling several "variable tables". Some of these tables are subsequently used to determine how much memory to allocate for the run in the "A" array, the primary array in the model within which all prognostic variables and many other important quantities are stored. Most new array space which a user would need to add to the standard model for a specialized application would be added to the "A" array. The required code changes would be made to **rvtab.f** and **rcommons.h**. Details of adding new array space are given in Section 9.4.

Section 2.2.2 - the lib subdirectory

The lib subdirectory contains additional atmospheric model code plus basic mathematical routines commonly accessed by the model, preprocessing, and postprocessing routines.

rmcn.f , **rmcro.f**, **rmdg.f**, **rmflw.f**, and **rmutl.f**, contain the subroutines for the old bulk microphysics parameterization.

rname.f supplements its namesake in the model subdirectory.

rrad.f contains the radiation parameterization.

rfft.f , **rfvec.f**, **rnumr.f**, **rpolr.f**, **rutil.f**, and **rfvor.f** contain miscellaneous utility subroutines.

rsys.F contains system-dependent code which must be edited specifically for each computer installation on which RAMS is executed. This procedure is explained in Chapter 3.

Section 2.2.3 - the lib/libc subdirectory

dirs.c , **ramsutils.c**, and **sub_names.h** contain routines written in the C computing language. They deal mainly with I/O and dynamic memory allocation which are easier in C than in FORTRAN.

Section 2.2.4 - the isan subdirectory

The isan subdirectory contains source code files involved in the objective analysis of three-dimensional observational data and assimilation of these data into the model. Unlike previous versions of RAMS, ISAN no longer directly ingests data in the format stored at NCAR or NMC, or of arbitrary format. This ingestion step is first performed by other routines in the subdirectories under the dprep directory which then write the data to files in a standard RAMS format. ISAN then inputs those standard-format data files. Because the standard files can be examined directly, it is easy to inspect the data at this stage.

aobj.f contains subroutines which perform the Barnes objective analysis.

aplt.f also contains graphical subroutines, especially those which call NCAR graphics routines.

asgen.f contains subroutine ISAN_DRIVER, the top-level driver in ISAN. It calls the top-level subroutine for each of the three stages of the package.

askewt.f contains subroutines which do the skew-T plots.

astga.f contains subroutines comprising part of the graphical capabilities of ISAN.

astgb.f contains additional subroutines for the graphical features of the package.

asti.f contains subroutines that perform objective analysis of the observational data.

asti2.f contains the subroutines that input upper air sounding data and surface data to ISAN.

astp.f contains subroutines that input gridded pressure level data to ISAN.

avarf.f contains subroutines which create the "variable initialization files", a blend of the isentropic, σ_z, and surface analyses onto the model grids.

Section 2.2.5 - the dprep/ncar subdirectory

The dprep/ncar directory contains routines that read observational data (upper air soundings, land surface, ship, buoy, and gridded pressure level data) in the archive format used at the National Center for Atmospheric Research (NCAR) and write the data out to files in a standard RAMS format. The output data files are to be read in by ISAN routines. This extra dprep step makes checking of observational data values and formats much easier because the files it generates are in a simple ASCII format.

Makefile is a UNIX makefile which controls selection, compilation, and archiving object code for files in this dprep/ncar directory. Its use is described more fully in Chapter 4.

PREP_IN is an input file read by the program created from source code in this dprep/ncar directory that contains namelist data. These data mainly specify input and output file names, observation times, type of data, and geographic region to be processed. The variables in PREP_IN are described more fully in Chapter 6.

asgen.f contains the top level driver that directs all the reads and writes.

asti2.f calls routines that read NCAR format upper air and surface data.

astp.f calls routines that read NCAR format pressure level data.

prep_com.h contains some parameter statements and common blocks that are inserted into several subroutines contained in this dprep/ncar subdirectory.

rassim.f contains NCAR Data Support Services (DSS) I/O subroutines for accessing NMC gridded analyses, upper-air, land surface, ship, and buoy data.

Section 2.2.6 - the dprep/generic subdirectory

The dprep/generic directory contains routines that read observational data of arbitrary format and write the data out to files in a standard RAMS format. Formats for the input files must be specified by the user; routines in this directory are templates for that purpose. The output data files are to be read in by ISAN routines. This extra dprep step makes checking of observational data values and formats much easier because the files it generates are in a simple ASCII format.

Makefile is a UNIX makefile which controls selection, compilation, and archiving object code for files in this dprep/generic directory. Its use is described more fully in Chapter 4.

PREP_IN is an input file read by the program created from source code in this dprep/ncar directory that contains namelist data. These data mainly specify input and output file names, observation times, type of data, and geographic region to be processed. The variables in PREP_IN are described more fully in Chapter 6.

asgen.f contains the top level driver that directs all the reads and writes.

namelist.info is a documented version of PREP_IN that is not read by a program but is only for user information. Unlike RAMSIN in the model directory, PREP_IN uses a standard FORTRAN namelist reader, which does not allow comments.

prep_com.h contains some parameter statements and common blocks that are inserted into several subroutines contained in this dprep/generic subdirectory.

press_read.f reads generic pressure level data. The user must insert the appropriate code to read this data.

read_rawi.f reads generic rawinsonde data. The user must insert the appropriate code to read this data.

read_sfc.f reads generic surface data. The user must insert the appropriate code to read this data.

Section 2.2.7- the mrc/revu subdirectory

The mrc/revu directory contains source code files that are specific to the REVU postprocessing routines.

REVU_IN is an input file read by the program created from source code in this mrc/revu directory that contains namelist data. These data specify the datasets to access and several details on the plots to be made. The variables in REVU_IN are described more fully in Chapter 8.

revu_main.f contains the main program for revu and calls high level subroutines.

rbat.f contains high level subroutines for preparing plots.

rgrab.f contains routines to read model output files and interpolate to arbitrary points in space and time. These routines are not yet functional but will be a future capability.

Section 2.2.8- the mrc/post subdirectory

The mrc/post subdirectory contains several lower-level routines involved in different types of postprocessing of RAMS model output. Many of the routines are called by REVU.

RAMSfilelist.f identifies and opens one or more of a set of files. This utility is used, for example, by REVU to read analysis files output from a RAMS model simulation.

barbs.f contains routines that plot wind barbs.

charutils.f contains subroutines that perform various operations on character strings such as parsing.

cread_RAMS.f is a top-level routine to read an individual field from RAMS files.

defparse.f parses files containing default information in REVU.

dumpout.f selects a subregion of a two- or three-dimensional field for printing or writing in REVU.

eenviron.c contains C code that checks and sets various machine environmental variables.

error_mess.f prints error messages.

hutil.f contains utility routines involving the coordinate transforms used in RAMS.

hvlib.f extracts specified field variables from RAMS model output (analysis) files.

iplt.f reads RAMS model output (analysis) files and drives the plotting of a field.

mkmap.f calls routines to draw a geographic map in a plot domain.

plt.f contains routines to draw parts of a plot such as borders, tick marks, and labels.

rainit.f performs some initialization steps for REVU.

ramscode.f contains modified code from the RAMS model that computes various quantities.

ramsgks.f contains several modified NCAR Graphics routines.

rcolors.f contains color tables for plot jobs.

rvec.f contains a vector plot routine.

Section 2.2.9- the van subdirectory

The van subdirectory contains routines to produce NCAR graphics plots from RAMS model output. It is being replaced by the newer REVU utility.

VAN_IN is an input file that contains namelist data. These data specify the datasets to access and several details on the plots to be made. The variables in VAN_IN are described more fully in Chapter 8.

rafld.f contains subroutines which select the particular fields specified for visualization, computing any derived quantities which are not directly transferred from the atmospheric model.

rcnfa.f contains subroutine DRIVER, which is the top-level driver for VAN, and also includes routines which read the data files generated by the atmospheric model.

rplt.f contains the subroutines which access the desired cross sections of data from the atmospheric model fields, and call the appropriate graphical routines to perform the actual plotting.

van_main.f contains the main program for VAN.

Section 2.2.10 - the include subdirectory

The include subdirectory contains files which are inserted into many subroutines in source code directories model, van, and isan via 'include' statements.

ascoms.h sets up common blocks and parameter statements for insertion to several subroutines in the isan directory.

rconfig.h contains a PARAMETER statement defining the dimensioning parameters of MAXGRDS, NXPMAX, NYPMAX, NZPMAX, NZGMAX, MAXSCLR, and MAXDIM. These are settings for the maximum values of the number of grids, grid points, and added prognostic scalar fields that may be used in a run. Actual computer central memory usage is not very dependent on these settings (since most memory is dynamically allocated), so these may be set fairly large. More details of what these variables represent are given in Chapter 7.

rcommons.h defines a series of parameter statements and common blocks which are used in more than half the subroutines of the model. This is nonexecutable code, which is easily inserted at the beginning of any subroutine with a single "include" statement.

rconstants.h contains a parameter statement with several physical constants.

vcommons.h defines some common blocks and parameter statements which are inserted into several subroutines in the van directory.

vcomm2.h defines some common blocks and parameter statements which are inserted into several subroutines in the van directory.

Section 2.2.11 - the bin subdirectory

The bin subdirectory initially contains only the master Makefile used for model installation, the additional makefiles Make.rams, Make.lib, Make.revu, and Make.van for recompiling user-modified files from the model, lib, revu, and van subdirectories, respectively, and a file called bat which is useful for submitting batch jobs. When the master Makefile is executed, it directs compilation and archiving of source code in the model, isan, van, revu, post, and lib subdirectories. This produces four archive files (RAMS_3b.a, RAMS_van.a, Revu.a, and librams.a) and three executable files (RAMS_3b, RAMS_van, and Revu), all of which are written to the bin directory.

Makefile is a UNIX makefile which controls selection, compilation, and archiving object code for files in the model, isan, lib, libc, van, and revu subdirectories. It is the master makefile, used for installing and building the main components of RAMS in a single step. Its use is described more fully in Chapters 3 and 4.

Make.rams is a UNIX makefile which controls selection, compilation, and archiving object code for files in this model directory. It is intended only for re-compilation of selected files that have been modified by the user. Its use is described more fully in Chapter 4.

Make.lib is a UNIX makefile which controls selection, compilation, and archiving object code for files in this lib subdirectory. It is intended only for re-compilation of selected files that have been modified by the user. Its use is described more fully in Chapter 4.

Make.revu is a UNIX makefile which controls selection, compilation, and archiving object code for files in this mrc/revu subdirectory. It is intended only for re-compilation of selected files that have been modified by the user. Its use is described more fully in Chapter 4.

Make.van is a UNIX makefile which controls selection, compilation, and archiving object code for files in this van subdirectory. It is intended only for re-compilation of selected files that have been modified by the user. Its use is described more fully in Chapter 4.

Section 2.2.12 - the util subdirectory

The util subdirectory contains a generic routine for the user to generate standard RAMS surface datasets such as topography, and an optional file of dummy NCAR graphics calls for those who do not have NCAR graphics.

mktopo.f used to convert topography, land percent, or sea-surface data to a specific format recognized by RAMS.

ncar_dummy.f contains dummy routines which allows the model to load and execute if your system does not support NCAR Graphics. Adding this file to the file list in the Makefile in the bin subdirectory, or if recompiling, in the Make.rams file in the model subdirectory enables the job to run without NCAR graphics.

Section 2.2.13 - the data/micro subdirectory

The data subdirectory contains several files read by the RAMS model for use by the new bulk microphysics code. These files are described in Section 7.4 in the descriptions of namelist variables COLTABFN and EVPTABFN.

Section 2.2.14 - the data/elev10m subdirectory

The data/elev10m subdirectory contains a global elevation dataset defined on a 10 arc-minute size mesh. Each of the several data files covers a 20-degree by 20-degree region of the earth and the latitude and longitude of the southwest corner of each region is encoded in the file name. A HEADER file for the dataset is also in the directory.

Section 2.2.15 - the data/pctl10m subdirectory

The data/pctl10m subdirectory contains a global land percentage dataset defined on a 10 arc-minute size mesh. Each of the several data files covers a 20-degree by 20-degree region of the earth and the latitude and longitude of the southwest corner of each region is encoded in the file name. A HEADER file for the dataset is also in the directory.

Section 2.2.16 - the data/sst subdirectory

The data/sst subdirectory contains a global climatological mean monthly sea surface temperature dataset defined on a 1-degree size mesh. A total of 24 data files, one for each hemisphere (east or west) and month are in the directory. The latitude and longitude of the southwest corner of each region (always 90 south) is encoded in the file name. A HEADER file for each month of the dataset is also in the directory.

Section 2.2.16 - the data/sample subdirectory

The data/sample/model_output subdirectory contains the standard output, standard error, and analysis files generated from a RAMS model sample run. The data/sample/revu_output subdirectory contains a metacode file generated from REVU of plotted output from the sample run. These are available for comparison and testing the success of a RAMS installation.

CHAPTER 3 - INSTALLATION OF RAMS

The installation of RAMS is quite easy. The following instructions are for the UNIX operating systems and its derivatives. If an operating system other than UNIX is to be used, these instructions and the Makefiles will need to be translated.

Installation now produces archive files for all major parts of RAMS, not just the library as before, and also produces executable files. It is suggested that in environments where several users will access RAMS on the same machine or other machines of the same type, the installation be done in a directory that is exported to all those users. Then everyone can easily access the archive files and need not recompile all the code. Even when users modify some of the code, they may still begin from the central installed archive files and need only recompile the modified files. This procedure is described in detail in Chapter 4.

The following steps should be carried out for installing RAMS:

1. Read the distribution media onto the directory where RAMS is to be installed. Please see separate instructions for the procedure.

2. Check directory structure. All directories and files listed in Section 2.2 should exist.

3. Examine **rsys.F** in the lib directory. This file contains all the machine-dependent features that RAMS uses. Each routine in this file must be examined and potentially modified for compatibility with your machine. Even if your machine is listed, there may be installation-specific parameters that may need to be changed (for example, mass storage system calls). There are several machines already supported in the file. The usual technique is to use one of the existing machines as an example and modify its parts for a new machine. Note the syntax of the C preprocessor for checking which machine has been defined. (For all of us "FORTRAN-centric" people, the machine was defined with a -D option on the command line in the script file. The || is equivalent to the FORTRAN .OR.) If you needed to use a different machine name for C_MACH other than the ones which were listed, make sure that each of the routines in **rsys.F** checks for your new machine name.

Following is a list of the routines in **rsys.F** and an explanation of each.

IOALLOW	Returns a 'Y' if a machine is allowed to do disk I/O. Most workstation class machines are brought to their knees if this is done. Larger machines such as CRAYs can do this without a great deal of inefficiency. If an 'N' is returned, the model will stop if the I/O scheme is started. The disk I/O feature is rather obsolete, a holdover from the olden days when the "big" machines (like CRAY-1's) had a whopping 8,000,000 bytes (count them, a whole 8 Mb!) of central memory. This feature really should not be used.
TIMING	Returns CPU time. Called once at beginning of timestep with ICALL=1 and at end with ICALL=2, then returned values are subtracted.
RAMRAN	Returns a random number. Internal parameter IRANDM is the random number table "seed".
INTRSHFT	Bit-manipulation. Returns the result of shifting IWORD to the right NSHFT bits. (The shift may be cyclic but this is not required.)
INTLSHFT	Bit-manipulation. Returns the result of shifting IWORD to the left NSHFT bits. (The shift may be cyclic, but this is not required.)

INTAND Bit-manipulation. Returns the result of a bit-by-bit "and" between IWORD1 and IWORD2.

INTOR Bit-manipulation. Returns the result of a bit-by-bit "or" between IWORD1 and IWORD2.

INTLMSK Bit-manipulation. Returns a word with a "mask" (bits set to 1) of NBITS on the left side of the word.

INTRMSK Bit-manipulation. Returns a word with a "mask" (bits set to 1) of NBITS on the right side of the word.

JCL Performs various file manipulations.

ENTRY JCLOPEN Opens a new file with a filename of FILENM, assigns it to unit number IUNIT with format of FORMT (either 'FORMATTED' or 'UNFORMATTED').

ENTRY JCLGET Accesses an existing file with the file name of FILENM and assigns it unit number IUNIT. For some machines, this routine will access the file from a mass storage system.

ENTRY JCLDISP Closes or disposes an attached unit with the local file name of FILENL to the permanent file name FILENP. Without mass storage systems, this is a simple CLOSE statement.

LOCFILE Returns a 'Y' if a machine needs a short (7 character) file name. Return 'N' otherwise. This will probably be 'N' for all machines but mainframes at supercomputer installations.

ASSUNT Assign a scratch random access file. Not needed unless disk I/O is to be done.

VREAD Read a record from the random access unit. Not needed unless disk I/O is to be done.

VWRITE Write a record to the random access unit. Not needed unless disk I/O is to be done.

4. In the "lib/libc" directory is a file called **sub_names.h**. In this file are C preprocessor statements that define the syntax of the names that the C routines need if they are to be called from FORTRAN. As you can see, this is not yet standard from machine to machine. However, we have yet to come across a machine where this cannot be done rather easily. There are only three variations that we have encountered. Let's take, for example, the routine "iralloc". The call of this function from a FORTRAN program on any machine would look like

```
iaddress=iralloc(numwords,array,ioffset)
```

However, the C routine function name needs to have a different syntax on different machines. On an IBM or HP workstation, the function name just needs to be in lower case:

```
iralloc(memtot,ia,ioff)
```

On a Sun, Alpha, or Silicon Graphics workstation, the name needs to be in lower case with an appended underbar:

```
iralloc_(memtot,ia,ioff)
```

and on a CRAY or the old Stardent TITAN machines, the name needs to be in upper case without the underbar.

```
IRALLOC(memtot,ia,ioff)
```

The file **sub_names.h** is "#include"d in the other C files. The C preprocessor will change the names in the actual code depending on which machine was defined in the **Makefile** (described below and in Chapter 4). If you are adding a new machine name, see if the FORTRAN-calling-C convention is the same as on the existing machines. If it is, just copy one of the #if/#endif blocks and change the machine name. If the convention is different, copy one of the blocks, change the machine name, and make the necessary changes to the *second* name of the function on each of the #define lines.

5. Read Section 7.1 of the User's Guide which describes parameters contained in the file rconfig.h in the **include** subdirectory. Determine the approprate values for these parameters for your installation and set them in the rconfig.h file.

6. Examine the **Makefile** in the bin subdirectory, which contains commands for building the major parts of RAMS. The installer should read Chapter 4 which gives describes in detail the contents of the Makefile.

Following are the things in the **Makefile** that may need to be modified:

C_MACH	The name of the machine on which you are compiling the RAMS software. If your machine is not in the list and you cannot use one of the listed machines without modifications in "rsys.F" (see item 3. above), add a name to this list and make the changes to "rsys.F" and "sub_names.h" as detailed above.
RAMS_ROOT	Define the full path of the rams3b directory for this variable.
NCARGF77	The name of the NCAR Graphics compile and loading script on your machine. If you do not have NCAR graphics, this should be set to the name of the FORTRAN compiler on your machine.
FORT_COMP	The name of the FORTRAN compiler on your machine.
FORT_OPTS	Any flags for the FORTRAN compiler you want to use, including optimization, etc.
C_COMP	The name of the C compiler on your machine.
C_OPTS	Any flags for the C compiler you want to use. Optimization is not required since the C routines only do memory allocation and I/O functions.
C_PP_OPTS	The flag for the C compiler which will only call the C preprocessor on the source file. This is used in "rsys.f", a FORTRAN source file.
ARCHIVE and RANLIB	These two commands need to accomplish the archiving of object files into an archive file (with a "**.a**" suffix) and creating a symbol table within the archive file. Some machines (CONVEX, SUN, maybe others) need a separate command to create the symbol table, so a two step process has been implemented for all machines. For many computers, the second step could be

	a dummy step (like an **ls** command, for example). Please check the Makefile comments for examples for a specific machine.
RAMS_SOURCE	If you do not have NCAR graphics on your machine, add the file $(RAMS_ROOT)/util/ncar_dummy.f to the list of files defined for RAMS_SOURCE.

End the editing session and make sure that Makefile has execute permission, giving, if necessary, the UNIX command:

```
chmod a+x Makefile
```

To execute the Makefile, type the command:

```
make
```

The Makefile carries out instructions for accessing and compiling source code files in the model, lib, libc, isan, revu, post, and van subdirectories, archiving the resultant object files, and generating executable files. Four archive files (RAMS_3b.a, RAMS_van.a, Revu.a, and librams.a) and three executable files (RAMS_3b, RAMS_van, and Revu), are generated from this procedure and are all written to the bin directory.

7. All files in or beneath the data subdirectory need to be uncompressed before they can be read by the model executable. To perform this task for each directory, cd to the directory and give the UNIX command:

```
uncompress *
```

These are the only steps necessary for installation of RAMS components before an actual run is made. See Chapter 4 for the procedures for making the actual runs.

CHAPTER 4 - RUNNING RAMS JOBS UNDER UNIX

Section 4.1 - The Makefile

Our standard method for running the RAMS software on a UNIX machine is to execute a Makefile which contains a series of UNIX commands to carry out the necessary steps. There are seven such Makefiles that exist in the directories: (1) Makefile in the **bin** directory is for initial installation of RAMS and controls the compiling and building of the source code in the model, lib, libc, isan, revu, post, and van directories; (2-5) Make.rams, Make.lib, Make.revu, and Make.van in the **bin** directory each control the recompilation and rebuilding of any files in model, lib, revu, and van directories, respectively, that require user-modification; and (6-7) Makefile in both the **dprep/ncar** and **dprep/generic** directories control compilation and building of all source code in those directories. All of these Makefiles have the same basic structure which is described below.

The main processes driven by the Makefile are: (1) define several shell variables which potentially contain machine-dependent parameters; (2) compile specified FORTRAN and possibly C souce code; (3) archive the object files to create one or more archive files; and (4) load the archive(s) with other libraries such as the RAMS library to generate one or more executables.

Following is a more detailed description of each part of the Makefiles. As an example, the Makefile in the **bin** directory will be used since it contains the most parts. This section will be organized so that a section of the Makefile will be followed by a brief narrative.

```
#
#         Define site and machine specific parameters.
#
#        Machine type:
#                        'IBM'        (RS/6000)
#                        'STARDENT'   (STARDENT Titan)
#                        'SUN'        (SUN Sparc)
#                        'HP'         (HP Model 7xx)
#                        'ALPHA'      (DEC ALPHA)
#                        'SGI'        (Silicon Graphics)
#                        'VAX'        (DEC VAX/VMS)
#                        'CRAY'       (CRAY/UNICOS)
#                        'CONVEX      (CONVEX/UNICOS)
C_MACH= IBM
```

C_MACH defines the name of the machine on which you are compiling the RAMS software. This information is used in the file rsys.F in the **lib** subdirectory and by ramsutils.c in the **lib/libc** subdirectory (see also Chapter 3. The list of available machine types is given in the commented lines (the comment character for the Makefile is the # sign).

```
RAMS_ROOT=/usr/local/rams3b

INCLUDES=$(RAMS_ROOT)/include
MODEL=$(RAMS_ROOT)/model
LIB=$(RAMS_ROOT)/lib
POST=$(RAMS_ROOT)/mrc/post
VAN=$(RAMS_ROOT)/van
REVU=$(RAMS_ROOT)/mrc/revu
ISAN=$(RAMS_ROOT)/isan
```

RAMS_ROOT Defines the full path of the top level rams3b directory.

INCLUDES, MODEL, LIB, POST, VAN, REVU, and ISAN define full paths to other directories below rams3b with the help of RAMS_ROOT defined above.

```
NCARGF77=ncargf77
FORT_COMP= f77
FORT_OPTS= -c -O -NA18000 -NQ30000 -qcharlen=10000
FORT_INCS= -I$(INCLUDES)

C_COMP= cc
C_OPTS= -O -I$(LIB)/libc
C_PP_OPTS= -P
```

NCARGF77 The name of the NCAR Graphics compile and loading script on your machine. If you do not have NCAR graphics, this should be set to the name of the FORTRAN compiler on your machine.

FORT_COMP The name of the FORTRAN compiler on your machine.

FORT_OPTS Any options (such as optimization, etc.) that should be on the compiler command.

FORT_INCS Relative path from top level RAMS directory of the directory containing all include files.

C_COMP The name of the C compiler on your machine.

C_OPTS Any flags for the C compiler you want to use. Optimization is not required since the C routines only do memory allocation and I/O functions.

C_PP_OPTS The flag for the C compiler which will only call the C preprocessor on the source file. This is used in "rsys.f", a FORTRAN source file.

```
#     For IBM,HP,SGI,STARDENT,ALPHA use these:
ARCHIVE=ar rs
RANLIB =ls
#     For SUN,CONVEX
#ARCHIVE=ar r
#RANLIB =ranlib
#     For HP
#OTHER_LIBS=-lc -lcl -lm
```

ARCHIVE and RANLIB are two commands that accomplish the archiving of object files into an archive file (with a "**.a**" suffix) and creating a symbol table within the archive file. Some machines (CONVEX, SUN, maybe others) need a separate command to create the symbol table, so a two step process has been implemented for all machines. For many computers, the second step could be a dummy step (like an **ls** command, for example). Please check the Makefile comments for examples for a specific machine.

OTHER_LIBS — defines other libraries that may need to be loaded on certain machines.

```
#
#     Define library archive and  model, VAN, and REVU
#            archive and executable file names.
#
LIBRAMS=librams.a

RAMS_EXE=RAMS_3b
RAMS_ARC=RAMS_3b.a

REVU_EXE=Revu
REVU_ARC=Revu.a

VAN_EXE=RAMS_van
VAN_ARC=RAMS_van.a
```

LIBRAMS — specifies the file name of the RAMS compiled library archive.

RAMS_ARC, REVU_ARC, and VAN_ARC are statements that specify the names, RAMS_3b.a, Revu.a, and RAMS_van.a, of three of the four archive files produced by this Makefile. The archive file must have the extension of ".a",

RAMS_EXE, REVU_EXE, and VAN_EXE are statements that specify the names, RAMS_3b, Revu, and RAMS_van, of the three executable files produced by this Makefile. The executable can be any name acceptable to your machine.

```
#
#     Define all necessary RAMS model source files
#
RAMS_SOURCE= \
$(ISAN)/asgen.f   $(ISAN)/astp.f   $(ISAN)/asti.f   $(ISAN)/aobj.f   \
$(ISAN)/asti2.f   $(ISAN)/astga.f  $(ISAN)/astgb.f  $(ISAN)/avarf.f  \
$(ISAN)/aplt.f    $(ISAN)/askewt.f  $(POST)/ramsgks.f \
$(MODEL)/rvtab.f  $(MODEL)/rdriv.f  $(MODEL)/rinit.f  \
$(MODEL)/rhhi.f   $(MODEL)/rvari.f  \
$(MODEL)/rnest.f  $(MODEL)/rtimh.f  $(MODEL)/radvl.f  \
$(MODEL)/radvf.f  $(MODEL)/raco.f   $(MODEL)/rhyd.f   \
$(MODEL)/rthrm.f  $(MODEL)/rturb.f  $(MODEL)/rbnd.f   \
$(MODEL)/rsurf.f  $(MODEL)/rconv.f  $(MODEL)/ruser.f  \
$(MODEL)/rmisc.f  $(MODEL)/rio.f    $(MODEL)/rcio.f   \
$(MODEL)/rname.f  $(MODEL)/rprnt.f  $(MODEL)/rusgs.f  \
$(MODEL)/rmicr.f  $(MODEL)/rmcth.f


#
#        Define REVU source files
#
REVU_SRC= \
        $(REVU)/rbat.f $(REVU)/rgrab.f  $(POST)/rvec.f \
```

```
$(POST)/mkmap.f $(POST)/iplt.f \
$(POST)/rainit.f $(POST)/hvlib.f $(POST)/ramscode.f \
$(POST)/plt.f $(POST)/ramsgks.f $(POST)/error_mess.f \
$(POST)/RAMSfilelist.f $(MODEL)/rcio.f \
$(POST)/cread_RAMS.f $(POST)/dumpout.f \
$(POST)/barbs.f $(POST)/defparse.f \
$(POST)/hutil.f $(POST)/rcolors.f $(POST)/charutils.f

#
#        Define VAN source files
#
VAN_SRC= \
        $(VAN)/rcnfa.f      $(VAN)/rplt.f    $(VAN)/rafld.f    \
$(POST)/ramsgks.f   $(POST)/rainit.f $(POST)/RAMSfilelist.f \
$(POST)/charutils.f $(POST)/ramscode.f


#
#        Define library source files
#
F_SOURCE = \
        $(LIB)/rfft.f     $(LIB)/rfvec.f    $(LIB)/rmcn.f  \
$(LIB)/rmcro.f    $(LIB)/rmdg.f     $(LIB)/rmflw.f \
$(LIB)/rmutl.f    $(LIB)/rname.f    $(LIB)/rnumr.f \
$(LIB)/rpolr.f    $(LIB)/rrad.f     $(LIB)/rutil.f \
        $(LIB)/rvfor.f

FC_SOURCE= $(LIB)/rsys.F
C_SOURCE= $(LIB)/libc/dirs.c       $(LIB)/libc/ramsutils.c



#==============================================================================
```

The purpose of this section is to define a few groupings of source code files for compilation. Each filename is preceded by a variable that contains the full path name, as defined above. A list of files is stored in each of the variables RAMS_SOURCE, REVU_SRC, VAN_SRC, F_SOURCE, FC_SOURCE, and C_SOURCE for convenience. Note that the backslash character at the end of a line indicates that the following line will be a continuation of the present line. No character, not even a blank, may follow the backslash on the same line.

```
all: $(LIBRAMS) $(RAMS_EXE) $(REVU_EXE) $(VAN_EXE)

#==============================================================================

$(LIBRAMS):: $(F_SOURCE)
        $(FORT_COMP) $(FORT_OPTS) $(?)
        $(ARCHIVE) $(LIBRAMS) $(?F:.f=.o)
        $(RANLIB) $(LIBRAMS)
        echo Finished building FORTRAN part === $(LIBRAMS)
        rm -f $(?F:.f=.o)

$(LIBRAMS):: $(FC_SOURCE)
        cp $(?) $(?F:.F=.c)
        $(C_COMP) $(C_PP_OPTS) -D$(C_MACH) $(?F:.F=.c)
        mv $(?F:.F=.i) $(?F:.F=a.f)
```

```
        $(FORT_COMP) $(FORT_OPTS) $(?F:.F=a.f)
        $(ARCHIVE) $(LIBRAMS) $(?F:.F=a.o)
        $(RANLIB) $(LIBRAMS)
        echo Finished building FORTRAN/C part === $(LIBRAMS)
        rm -f $(?F:.F=a.o) $(?F:.F=a.f) $(?F:.F=.c)

$(LIBRAMS):: $(C_SOURCE)
        $(C_COMP) -c $(C_OPTS) -D$(C_MACH) $(?)
        $(ARCHIVE) $(LIBRAMS) $(?F:.c=.o)
        $(RANLIB) $(LIBRAMS)
        echo Finished building C part === $(LIBRAMS)
        rm -f $(?F:.c=.o)


#===============================================================================

$(RAMS_EXE): $(RAMS_ARC)
        $(NCARGF77) -o $(RAMS_EXE) $(MODEL)/rammain.f $(RAMS_ARC) \
        $(LIBRAMS) $(OTHER_LIBS)
        echo Finished building === $(RAMS_EXE)
        rm -f rammain.o

$(RAMS_ARC): $(RAMS_SOURCE)
        $(FORT_COMP) $(FORT_OPTS) $(FORT_INCS) $(?)
        $(ARCHIVE) $(RAMS_ARC) $(?F:.f=.o)
        $(RANLIB) $(RAMS_ARC)
        rm -f $(?F:.f=.o)



#===============================================================================

$(REVU_EXE): $(REVU_ARC)
        $(NCARGF77) -o $(REVU_EXE) $(REVU)/revu_main.f $(REVU_ARC) \
        $(LIBRAMS) $(OTHER_LIBS) $(FORT_INCS)
        echo Finished building === $(REVU_EXE)
        rm -f revu_main.o

$(REVU_ARC):: $(REVU_SRC)
        $(FORT_COMP) $(FORT_OPTS) $(FORT_INCS) $(?)
        $(ARCHIVE) $(REVU_ARC) $(?F:.f=.o)
        $(RANLIB) $(REVU_ARC)
        rm -f $(?F:.f=.o)


#===============================================================================



$(VAN_EXE): $(VAN_ARC)
        $(NCARGF77) -o $(VAN_EXE) $(VAN)/van_main.f $(VAN_ARC) \
        $(LIBRAMS) $(RAMS_ARC) $(OTHER_LIBS) $(FORT_INCS)
        echo Finished building === $(VAN_EXE)
        rm -f van_main.o

$(VAN_ARC):: $(VAN_SRC)
        $(FORT_COMP) $(FORT_OPTS) $(FORT_INCS) $(?)
        $(ARCHIVE) $(VAN_ARC) $(?F:.f=.o)
```

```
$(RANLIB) $(VAN_ARC)
rm -f $(?F:.f=.o)
```

The above section of the Makefile directs the actual compilation, archiving, and building of the code, making use of several variables defined earlier in the Makefile. This particular Makefile from the **bin** directory, which is designed to facilitate installation of RAMS, produces four archive and three executable files and writes them to the **bin** directory. These archive and executable files are the intended starting point for the general user running RAMS code either with or without modifying code.

The four supplementary makefiles: Make.rams, Make.lib, Make.revu, and Make.van, are easier to use for recompiling modified code. Each is a proper subset of the Makefile in the bin directory, geared toward compiling and building only the code in its respective directory, except that the source code file list is, by default, commented out. The user inserts only the filename(s) that have been modified. The makefile automatically creates a modified archive from the standard installation and the modified source code file.

The Makefiles in the **dprep/ncar** and **dprep/generic** directories are also very similar to, but smaller than, the example shown above. They are used only to compile the code files in their respective directories. The **generic** directory in particular contains code that always must be modified by the user which is why it is not compiled as part of the larger RAMS model compilation.

The Makefile will maintain an archive of compiled object files (see **ramsarc** below) so that all the code does not need to be compiled on every run. Here are some basic rules:

- The first time that a particular configuration is run, all the code needs to be compiled. Normally this is accomplished in installing and building RAMS, described in Chapter 3. On subsequent runs of the same configuration, assuming the compiled archive file still exists, only those files that are changed need to be compiled. The Makefile determines which files require recompilation automatically by comparing the date and time of the last update to the model code file and the object file of that code file that is contained in the archive. Only code files that are newer than the archived object file are re-compiled.

- If a file needs to be changed and recompiled for a specific application (that is, not a permanant change like a bug fix that would be made to the standard code directories), copy the file from the standard RAMS directory to a working directory. Also copy the standard makefile from that directory (for example, if the file to be changed is in the standard **model** directory, copy Make.rams to the working directory). In addition, copy the appropriate archive file from the standard **bin** directory to the working directory (in the preceding example where part of the model, not van or revu, is being recompiled, copy the archive file RAMS_3b.a). Edit the copied Make.rams so that it lists the changed source code file and its correct directory following one of the commented examples given. Before compilation, make sure that the modified source code file has a later date than the copied archive file. This can be ensured by giving the command "touch" followed by the source file name. Then execute Make.rams (the UNIX command to do this is "make -f Make.rams" since the file name is not the default Makefile). This will put object code from the modified source code file in the copied archive, replacing the original code from the standard file. It will also produce an executable file in the working directory.

- If after having compiled a changed module, one needs to return to the standard version of that module (in the above example, to return to using ruser3b.f from the standard model directory), simply editing the Makefile so as to restore the path to the standard directory for that file will not cause the standard file to be recompiled. This is because the standard file will normally have an older date than the archive. Even if one deletes the object of that file from the archive, it will not be recognized that the file needs to be compiled and re-added to the archive. One could of course simply change the date of the standard version of the file by giving the UNIX command "touch" followed by the filename, but that would cause the file to be recompiled for everyone's application that accesses the standard model directory. In this case, it is better either to just copy the standard archive back to the working directory, or to copy the standard file to the working directory, make sure it has a date more recent than the working directory archive, and recompile it there.

- After giving the "make" command, which executes the Makefile, never kill the compilation process by giving the control-C command. This sometimes destroys the entire archive. If one realizes while compilation is in progress that an error was made and that compilation will have to be done over, one may either wait for the first compilation to finish, or if one is in a big hurry, one can give the control-Z command to suspend the first compilation, copy the pre-existing archive to a new file, resume the first compilation with the UNIX "fg" (foreground) command, kill the first compilation with the control_C command (resulting in the loss of the pre-existing archive, copy the saved archive back to the original name that was just lost, and finally begin the second compilation after the error is fixed. This sounds involved, but you get used to it after awhile.
- Tab characters, not blank characters, are used to indent continuation lines in the Makefile. This is a requirement.

If the user does not need to modify any source code, all that is required is to copy the executable (e.g., RAMS_3b) from the standard **bin** directory to the working directory and make sure it has execute permission. If it does not, give the command "chmod a+x RAMS_3b". Alternatively, the standard archive (e.g., RAMS_3b.a) and the corresponding makefile (e.g., Make.rams) may be copied to the working directory, followed by executation of the copied Make.rams. This will generate an executable in the working directory.

Section 4.2 - Running the Executable

Once the executable (e.g., RAMS_3b) is in the working directory in the desired form, it is ready for execution. Before this is done, a copy of the main input data file for the program (i.e., RAMSIN for the model, REVU_IN for Revu, VAN_IN for van, and PREP_IN for either set of prepmod routines) must be copied to the working directory and modified according to the specifics of the intended run. This procedure is described for the various input data files in Chapters 6-8.

Several options for running the executable are as follows:

1. You can give the name of the executable file as a command on the command line. This will execute the model in the current shell. In conjunction with this, other things can be done such as redirecting standard output and standard error to files and/or putting the job in background. See your UNIX manuals or man pages for details.
2. Check into the use of UNIX commands such as **nice**, **nohup**, etc.
3. Some machines (CRAY, for example) have batch subsystem software such as NQS. See your machine's documentation.
4. A common method on machines without batch subsystems is to use the standard UNIX "batch" facility. Create another small script that will take the RAMS script file and submit it to the batch queue on your machine. For instance, create a file called *bat* with execute permission which looks like this:

```
batch << !
$1 1> $2 2> $3
!
```

To use this script, put the **bat** file in a directory that is in your **$PATH**. Then type, for example, *bat* **RAMS_3b 3b.lis 3b.err**, where **RAMS_3b** is the name of the executable you want to run, **3b.lis** is the file name where standard output (the regular listing file from the run) is put, and **3b.err** is the file name where various messages from the operating system and compilers (e.g., messages about files that failed to compile) are put.

5. Finally, create a script called **run** which contains the following.

```
batch << !
$1.job 1>$1.list 2>$1.err
!
```

   You may want to put **run** in a place where you may have other scripts, such as a **bin** directory under your home directory. You can now run the model simply by typing

```
run r
```

   The job will be put into batch mode, and will create two files, **r.err** which contains the standard error messages as well as information about the CPU time used (specifics depend on the system used) and **r.list** which gives you the standard output.

If you run the model without having recompiled, for instance, a series of sensitivity tests where only namelist variables are changed, you simply need to edit RAMSIN and then rerun the model. Use the *cp* command to make a copy of RAMSIN (e.g., RAMSIN.run1, RAMSIN.run2, etc.) if you need a record of the namelist settings for each run.

Section 4.3 - Setting up and executing the sample run

A sample run has been set up so that you can do a simple simulation and compare your results with the same run we did on an IBM RS/6000 workstation. This allows you to check your installation of RAMS. The description of this simulation also provides an example of how to set up a model run and examine the results. Before going into details, we will take an overview of what is in subdirectory **sample**.

Subdirectory **sample** consists of two subdirectories **model_output**, which contains the standard output, standard error, and analysis files from the sample run, and **revu_output** which contains a metacode file created by executing REVU on these analysis files. The sample model run is a 12-hour simulation from 12 UTC to 00 UTC in June on a single-grid domain that covers the Florida Peninsula and some of the surrounding water. A sea-breeze readily develops and penetrates inland from all coastlines. No code modifications from the standard model installation are required to perform this simulation, and the RAMS model namelist file RAMSIN that is contained in the standard **model** subdirectory is the one used for this sample run.

We describe next the steps to follow in carrying out this sample run on your machine and comparing your results with those in the sample directory. We also explain many of the namelist variable settings in RAMSIN and REVU_IN that are relevant to the model simulation and to plotting its results. We will assume that you are working on the same type of computer with the same version of operating system as the computer on which the standard model installation has been made and that the standard model directories are accessible to you. (We recommend that a standard installation be done on each different type of computer and operating system version you or your group work with so that all subsequent model applications are easy to perform.)

Create a working directory where you will run the model and copy to it the RAMSIN file from the standard **model** directory. Edit the RAMSIN file to make any necessary pathname changes to files listed in it. The model namelist variables in RAMSIN that in general may require modified pathnames are HFILIN, HFILOUT, AFILOUT, VARFIL, SFCFILES, ITOPTFN, IPCTLFN, ISSTFN, IVEGTFN, COLTABFN, and EVPTABFN, all of which are in the **MODEL_FILE_INFO** namelist, but for this sample run, only HFILOUT, AFILOUT, ITOPTFN, IPCTLFN, ISSTFN, and IVEGTFN are used. Furthermore, of these, only ITOPTFN, IPCTLFN, ISSTFN, and IVEGTFN may require changing; they require the relative or full path from your working directory to the standard model data directory.

While the variables in RAMSIN are already set up to perform this sample run of the summertime Florida sea breeze, we shall go over them here to help illustrate the process of configuring a simulation. Many of the variables in RAMSIN are not relevant to this simulation, and we will confine discussion to those that are. More complete descriptions of all namelist variables in RAMSIN are given in Chapters 6 and 7, and the user should become familiar with these.

We first examine the **$MODEL_GRIDS** namelist. IOTYPE is set to 0 because we are doing a normal model run. RUNTYPE is set to 'INITIAL' because this run will initialize the simulation and continue from there. TIMEUNIT and TIMMAX values indicate that the run will be for 12 hours of simulation time. Only a single grid will be used (NGRIDS = 1), and the grid will have 40 points spanning the x-direction (which is essentially east-west), 30 points in the y-direction (north-south), 25 vertical levels, and 11 soil levels (the first value specified for NNXP, NNYP, NNZP, and NNZG). The horizontal grid spacing is set to 15 km (DELTAX and DELTAY). The vertical grid spacing begins at 75 m at the ground (DELTAZ) is stretched by a factor of 1.25 (DZRAT) in each consecutive model level upward, and is held to a maximum of 1 km (DZMAX) once stretching leads to large spacing. The model timestep (DTLONG) is set to 30 seconds. The calendar time of the beginning of the simulation is July 14, 1991 at 12 UTC (IMONTH1, IDATE1, IYEAR1, and STRTIM). Both the point of tangency (POLELAT and POLELON) of the model polar stereographic grid and the center (CENTLAT and CENTLON, first entry) of the grid are located at 28.0 degrees north latitude and 82.0 degrees west longitude. This location, the number of grid points, and the grid spacing define the model domain as covering most of the Florida peninsula plus some surrounding water.

Next we examine the **$MODEL_FILE_INFO** namelist. IOUTPUT is set to 1 in order to output history and analysis files from this run. We will not use the history files, but the analysis files will be used for plotting model output. The history and analysis file prefixes 'h.a' and 's.a' (assigned to HFILOUT and AFILOUT) will be the first part of all history and analysis filenames written. A history file will be written at intervals of 43200 seconds (FRQHIS), which will result in one file at the initialization time and one at the end of the simulation, and will have the write times expressed in seconds (HFUNITS) encoded in the file name. Analysis files will be written every 7200 seconds (FRQANL) and will also have the write times encoded in the file names in seconds (AFUNITS). This simulation will be initialized horizontally homogeneously (INITIAL = 1) from the sounding defined in the **$MODEL_SOUND** namelist below. Selected model fields in the **$MODEL_PRINT** namelist below will be written to the standard output file every 7200 seconds (FRQPRT). Also to the standard output file will be written a timestep message every timestep (ISTPFL = 1), selected model fields at the initial time (INITFLD = 1), and a list of RAMSIN namelist variables and their assigned values (INPRTFL).

We obtain topographic height, land percentage, and sea surface temperature from standard RAMS global data files (ITOPTFLG, IPCTLFLG, and ISSTFLG all set to 1), but vegetation type is assigned a value from the namelist (IVEGTFLG = 2). The filenames for topographic, land percentage, and sea surface temperature data are specified in ITOPTFN, IPCTLFN, and ISSTFN. This is where you need to insert the correct full or relative path to the rams3b/data directories from your working directory. We will do no silhouette averaging of topographic data (SILAVWT = 0.0), but will smooth it some (TOPTWVL = 4.0). Neither land percentage nor sea surface temperature data will be smoothed (PCTLWVL = 2.0, SSTWVL = 2.0). The land percentage data, along with the model domain size and location defined above, automatically represents those parts of the domain that are land and those that are water.

In the **$MODEL_OPTIONS** namelist, we use hybrid time differencing (ITMDIFF = 3) and the nonhydrostatic equation set (NONHYD = 1). We let the model compute its own sound speed (SSPCT = 0). A wall-on-top upper boundary condition (IBCTOP = 0) is used. At the lateral boundaries we use a Klemp and Wilhelmson condition (IBND = 1, JBND = 1) for the normal velocity component and a zero normal gradient condition (LSFLG = 0) on all other variables. Second order advection (IADVL = 2, IADVF = 2) and the Mahrer/Pielke radiation scheme (ISWRTYP = 2, ILWRTYP = 2) are selected with radiation fluxes calculated every 1200 seconds (RADFRQ) and longitudinal variation of sun angle accounted for (LONRAD = 1). No cumulus parameterization is used (NNQPARM = 0).

The prognostic soil and vegetation model are used (ISFCL = 1) with a vegetation type of 'Crop and Mixed Farming' (NVGCON = 1) and a soil type of 'sandy clay loam' (NSLCON = 6). The eleven soil model depths in meters are specified in SLZ, the soil moisture of each level in fraction of the maximum possible in SLMSTR, and the initial soil temperature relative to the initial low-level air temperature in STGOFF. Thus, the deepest soil layers are initially 40% moist and 5 K warmer than the air, while the shallowest layer is only 20% moist and 2 K colder than the air (this is reasonable for the model initialization time a little after sunrise in a situation where rain has not fallen recently).

The Mellor and Yamada and the Smagorinsky diffusion parameterization schemes are applied in the vertical and horizontal directions, respectively (IDIFFK = 1), with moderate horizontal diffusion scale length specified (CSX = .20, XKHKM = 3., AKMIN = 1.0). No condensation of moisture is considered in this simulation but water vapor is prognosed as a tracer quantity and for its effect on air density (NLEVEL = 1).

The **$MODEL_SOUND** namelist defines the single sounding from which the model atmosphere initial conditions are interpolated horizontally homogeneously. The sounding is defined by its heights, potential temperatures, relative humidities, and wind components in the namelist variables PS, TS, RTS, US, and VS, as instructed in namelist variables IPSFLG, ITSFLG, IRTSFLG, and IUSFLG. Sounding data up to a height of 25 km are specified although the model top is considerably lower. The potential temperature lapse rate is fairly low below 10 km, implying low static stability, with high stability above. Relative humidity is 60% up to 3 km and 10% above 3.5 km. A uniform easterly wind of 5 m/s is specified at all heights.

In the **$MODEL_PRINT** namelist, we specify 3 fields to be written to the standard output file: vertical velocity (WP), potential temperature (THP), and total water mixing ratio (RT). A horizontal cross section for each (IXSCTN = 3,3,3) at the lowest model level (ISBVAL = 2,2,2).

Most of the namelist variables that we have not mentioned, including all variables following the **$MODEL_PRINT** namelist, are not used in this sample run. The others not mentioned are not very important for the purpose of the sample.

Once any required changes have been made to the file pathnames in RAMSIN, the job may be run using the executable in the standard rams3b/bin directory. If you do not already have a batch job file such as **bat** described in Section 4.2 and located in the standard rams3b/bin directory, get or make one. You may simply copy this to your working directory or put one in a directory that is specified in your UNIX path list. To run the job, give the UNIX command:

```
bat PATH/rams3b/bin/RAMS_3b model.list model.err
```

where 'PATH' must be the full or relative path to the standard RAMS directory from your working directory. As described in Chapter 3, RAMS_3b is the standard model executable, which by the above command you will be executing from your working directory. The standard output and standard error from this run will be written to the files **model.list** and **model.err**, respectively, both in your working directory.

When the run is complete you should have in your working model directory all of the files contained in the standard sample directory, that is, the standard output file, the standard error file, the history files, and the analysis files, plus the RAMSIN and possibly bat files you originally copied there. Make sure that you have the same number of history and analysis files; fewer analysis files would indicate that the run did not finish successfully.

Files can be compared between your working directory and the standard model sample directory in a number of ways. One method is to use the UNIX *diff* command which gives differences between two specified files. The output from this command may be voluminous even if there are no numerically-significant differences between the files since different systems may have slightly different rounding which means results would not be identical. Direct side-by-side comparison between files using windows or hard copies is useful, particularly in combination with the **diff** command.

Examine your standard error file **model.err** for any runtime error messages. The standard error file in the **sample/model_output** directory is empty indicating no errors. Also examine your standard output file **model.list** which has a wealth of information from the model run. The first part of this file lists any warning messages from model subroutines 'OPSPEC1' and 'OPSPEC2' which check for reasonable namelist variable settings and gives memory usage statistics for the run. This section should be identical between your file and that in the standard sample directory. Next, the **model.list** file lists grid locations, information on the sound speed, the names of the analysis and history files written at the initial time for the run, the sounding and reference state profiles, and the values defined for namelist variables in RAMSIN. These should also be identical between your **model.list** file and the one in the **sample/model_output** directory, except for possible roundoff errors in floating point numbers printed with high precision, as mentioned above. Next in the file is a listing of topography height, land percentage, surface roughness, soil textural class, and sea surface temperature at each horizontal grid point. Because these are not printed at full precision, all or nearly all of these values should be identical between the files, but a few may differ in the last decimal place shown. Initial values of prognostic fields at the lowest model level (those specified in the **$MODEL_PRINT** namelist) are listed next. Again, these should be identical between the two files, except possibly for a few values that differ in the last decimal place due to roundoff error. One final set of error messages from subroutine 'OPSPEC3' follows, and is the last output from the model initialization process.

A series of messages, one line for each timestep, is the first output from the time integration. It is expected that the CPU times at the end of each line will be different, reflecting differences in execution time by the your computer and the IBM 550 RS6000 we did the sample run on. At 7200 seconds, the same selected prognostic fields are written. At this time, larger differences are expected between the sample run and yours since any differences between machine algorithms have been able to accumulate. Still, the differences should be relatively small and not qualitatively alter the simulation. Your results should reflect a sea-breeze of similar strength and location to that in the sample run. More timestep messages and prognistic field value prints alternate every 2 hours until the end of the run at 12 hours. As at 7200 seconds, the prognostic field values should be qualitatively similar between the two runs. Easier qualitative comparisons may be made between graphic plots of the results which are described below. A banner about writing analysis and history files finishes **model.list**.

Check your analysis file size which should be identical with the sample analysis file size since they are written in packed ASCII format (see Chapter 5). The analysis header files (those whose names begin with 's.a' and ending with 'head') may be compared directly with the **diff** command. Only full-precision floating point numbers should differ between the files, and these only in the last 1 or 2 decimal places.

Now we turn to plotting the results from the model runs. Create a new (REVU) working directory, preferably one immediately below your model working directory. Copy the file **REVU_IN** from the **mrc/revu** subdirectory to this directory. Edit your copy of **REVU_IN** and check path names specified for the namelist variable ANPREF so that it refers to the analysis files produced in your model run. If you have made your REVU working directory just below your working model directory, the relative paths already in **REVU_IN** should be correct.

We refer to Chapter 8 for descriptions of all namelist variables in **REVU_IN**. However, we point out here that IZTRAN is set to 1, ZVAR is set to '/F/2:2/', XVAR is set to '/H/0:0/', and YVAR is set to '/V/0:0/' so that the plots generated by REVU will be horizontal plots at the lowest model level, corresponding to the fields printed in the model standard output file **model.list**. We have set TVAR to '/F:7:7:1/' so that plots will be made from all 7 of the analysis files. In the CFRAME variables, we have instructed REVU to plot topographic height with wind vectors in the first frame potential temperature in the second frame, and total water mixing ratio in the third frame. A total of 21 frames are generated, 3 for each of the 7 analysis file times.

Run this REVU job by giving the UNIX command:

```
bat PATH/rams3b/bin/REVU revu.list revu.err
```

where 'PATH' must be the full or relative path to the standard RAMS directory from your working directory. As described in Chapter 3, REVU is the standard REVU executable, which by the above command you will be executing from your working directory. The standard output and standard error from this run will be written to the files **revu.list** and **revu.err**, respectively, both in your working directory.

In addition to the standard output and error files, a file named **gmeta** will be generated which contains the 21 frames of graphical output. These may be viewed with an appropriate metacode translator such as **idt** or **ictrans**. Compare each frame against those in the **gmeta** file in the **sample/revu_output** directory. Relatively small quantitative differences might be expected, but significant qualitative differences should not.

CHAPTER 5 - INPUT AND OUTPUT DATA FILES USED IN RAMS

Chapter 2 described the organization of RAMS into four main functional parts: (1) the model, (2) ISAN, (3) DATAPREP, and (4) REVU or VAN. Each of these parts requires input data files and produces output data files. This Chapter lists the input and output files for each of these components of RAMS, describing the form and purpose of each. Many of these files are optional, depending on the particular manner in which RAMS is used. For example, the simplest use of RAMS is to run the Atmospheric Model entirely from namelist input, and to generate only the Standard Output and Standard Error files from the model for examining model results. This would eliminate the use of ISAN, DATAPREP, and REVU or VAN.

Each of the above RAMS components will output at least the following two files on each run: 1) **Standard Output** is the standard file created by the operating system each time a job is submitted for execution. The name of this file is specified by the user when submitting the job. It contains information written to it by the operating system and compiler concerning the compilation, loading, and execution of the job, and also information written by the FORTRAN code concerning the operation of the programs, and 2)**Standard Error** is similar to the **Standard Output** file, but consists of any warning or error messages issued by the operating system or compiler. The name of this file is specified by the user.

Section 5.1 - Data Input to DATAPREP

Pressure-Level Data is a gridded dataset containing horizontal velocity components, temperature, and relative humidity as a function of pressure (the vertical coordinate). The standard spacing of these data obtained from the U.S. National Meteorological Center (NMC) or the European Center for Medium Range Forecasting (ECMWF) and archived at NCAR is 2.5 degrees in latitude and longitude on mandatory pressure levels (i.e., 1000 mb, 850 mb, 700 mb, etc.). DATAPREP in the **ncar** subdirectory is programmed to read these data in the format used at NCAR. Other gridded pressure coordinate datasets may be used with DATAPREP in the **generic** directory.

Surface Data consists of surface station reports of wind, temperature, humidity, and pressure, along with station location and elevation. DATAPREP in the **ncar** directory reads these data in the format archived at NCAR. Other surface datasets may be used with DATAPREP in the **generic** directory.

Rawinsonde Data from normally twice-daily rawinsondes. These data may be used to supplement the gridded Pressure Level Data. DATAPREP in the **ncar** directory reads these data in the format archived at NCAR. Other upper rawinsonde datasets may be used with DATAPREP in the **generic** directory.

Namelist Data are contained in the data file PREP_IN for either the **ncar** or **generic** versions of DATAPREP.

Note that the first 3 types of datasets are not part of the standard RAMS package, but must be acquired by the user.

Section 5.2 - Data Output by DATAPREP

Pressure Files contain gridded pressure level data in a specific RAMS format.

Rawindonde Files contain rawinsonde level data in a specific RAMS format.

Rawindonde Files contain surface data in a specific RAMS format.

Section 5.3 - Data Input to RAMS/ISAN

Pressure Files , **Rawinsonde Files**, **Surface Files** described in Section 5.2.

Namelist Data are contained in the data file RAMSIN in the **model** directory. These data specify many instructions for ISAN.

Section 5.4 - Data Output by ISAN

Pressure Files contain gridded data from either NMC or ECMWF which can be plotted for inspection via the ISAN graphics.

Isentropic Files consist of (1) gridded upper-air atmospheric data defined on the specified RAMS polar-stereographic grids at chosen isentropic levels, and (2) gridded surface data defined at the same horizontal coordinates. ISAN generates these files based on its analysis of the atmospheric data input to it. The Isentropic Files are used to create the "Variable files" or are used for independent analysis.

σ_z **Files** consist of gridded upper-air atmospheric data defined on the specified RAMS polar-stereographic grids at chosen σ_z levels. ISAN generates these files based on its analysis of the atmospheric data input to it. The σ_z Files are used to create the "Variable files" or are used for independent analysis.

Variable Files are files generated from ISAN which consist of data interpolated from the Isentropic Files and σ_z Files to the Atmospheric Model grid, and are input to the model in a run where namelist variable RUNTYPE is set to either 'INITIAL' or 'HISTORY'(see Chapter 7).

Section 5.5 - Data Input to the Atmospheric Model

Variable Files are described above in Section 5.4.

Terrain Height Data may be either a dataset which is now standard for RAMS, or it may be a special file input to the model by means of a read statement inserted by the user (see Section 9.1). The standard terrain height dataset is created by a file with the name **mktopo.f**. It generates a series of model-compatible terrain height files, each covering a latitude-longitude square of the earth, from user-supplied data.

Land Percent Data may be either a dataset which is now standard for RAMS, or it may be a special file input to the model by means of a read statement inserted by the user (see Section 9.1). The standard land percentage dataset is created by a file with the name **mktopo.f**. It generates a series of model-compatible land percentage files, each covering a latitude-longitude square of the earth, from user-supplied data.

Sea Surface Temperature may be either a dataset which is now standard for RAMS, or it may be a special file input to the model by means of a read statement inserted by the user (see Section 9.1). The standard sea surface temperature dataset is created by a file with the name **mktopo.f**. It generates a series of model-compatible sea surface temperature files, each covering a latitude-longitude square of the earth, from user-supplied data.

Vegetation Type Data may be either a dataset which is now standard for RAMS, or it may be a special file input to the model by means of a read statement inserted by the user (see Section 9.1).

Miscellaneous Data consisting of terrain height, land percentage, sea surface temperature, soil type, bare soil surface roughness length, soil temperature, soil moisture, vegetation temperature, vegetation moisture, and/or vegetation type may be input to the model by means of a read statement inserted by the user (see Section 9.1).

Namelist Data are contained in the data file RAMSIN. These data define the configuration of a run of the model, and control its options. The namelist data include sounding data which may optionally be used for initializing the model.

History Files are generated by the Atmospheric Model itself. They are used to re-start a model job, continuing the model solution from a point reached in a previous run. The History Files are written binary or in packed ASCII format, and contain sufficient information to restart the model with no sign that it was interrupted.

Section 5.6 - Data Output by the Atmospheric Model

History Files are described above in Section 5.5.

Analysis Files contain data from the Atmospheric Model prognostic fields plus additional information including the model grid configuration. They are used as input to REVU or VAN which produces visual depictions of the Atmospheric Model fields. These files are also used for other post-processing applications. The Analysis Files are written in packed ASCII format so they are still usable in cases where the Atmospheric Model and REVU or VAN are not executed on the same type of computer.

Section 5.7 - Data Input to REVU and VAN

Analysis Files are described above in Section 5.4.

Namelist Data are contained in the data files REVU_IN or VAN_IN. These data provide several instructions for REVU or VAN.

Section 5.8 - Data Output by REVU and VAN

Metacode File contains the graphical output from NCAR Graphics depicting the model fields. The metacode file may be translated for display on a computer monitor screen or for printing on a hardcopy device.

CHAPTER 6 - INGESTION AND OBJECTIVE ANALYSIS OF OBSERVATIONAL DATA: THE DATAPREP AND ISAN PACKAGES

Section 6.1 - Introduction

There are two basic methods in RAMS for initializing atmospheric fields: horizontally homogeneous initialization and variable initialization. The first method is very simple and consists of interpolating data from a single vertical sounding horizontally homogeneously onto the model grid(s). This option is specified in the RAMSIN namelist file by setting the INITIAL variable to 1 (see Section 7.3). The sounding data for this option is likewise defined by the user in RAMSIN (see Section 7.5).

The second method, which is much more involved and is the subject of this chapter, consists of blending data of one or more types from multiple locations to produce three-dimensionally variable initial model fields of velocity, temperature, pressure, and moisture. There are three principal steps to this procedure. The third step is the actual running of the model, which requires the INITIAL variable in RAMSIN to be set to 2, and requires *initialization files* to exist and to be read by the model (see description of VARFIL in Section 7.3). The first two steps together perform the input and blending of atmospheric data and produce the initialization files which contain the data interpolated to the model grid. When the model is run with variable initialization, it uses initialization files to supply not only the initial conditions but also the lateral boundary conditions at later times until the end of the run. Thus, initialization files need to be generated for two or more data times which bracket the time period covered by the run. Normally, initialization files are generated at 12-hour intervals, at the 00 UTC and 12 UTC times that upper air data are recorded.

The first step in generating initialization files is to run the stand-alone DATAPREP job(s) which access data in standard NMC or ECMWF formats (as archived at NCAR) and/or in arbitrary formats. This step simply reads the data and writes them to output files in a different (standard RAMS) format; details of the procedure are described below in Section 6.2. The second step is to run the ISentropic ANalysis package (ISAN) which inputs the data in the specific RAMS format, performs the data blending, interpolates the blended data to the model grid, and outputs the result to the initialization files. This procedure is described in Section 6.3.

Section 6.2 - Structure and Function of DATAPREP

DATAPREP is comprised largely of code that was formerly part of ISAN and performed the task of reading atmospheric data either in a standard format archived at NCAR or in other arbitrary formats. This code was moved to the separate, stand-alone job step in order to facilitate checking the result of this data ingestion. The main benefit is that DATAPREP outputs the data, with no processing other than discarding unneeded or bad data, to separate files written in a simple ASCII format. User familiarization with this format allows easy visual checking of data prior to processing by ISAN.

There are two separate DATAPREP jobs, one specifically for reading sounding and surface observations and 2.5-degree NMC or ECMWF gridded pressure level data in the format archived at NCAR, and the other in the form of a template to be modified by a user to input data in any format. Pressure level data would normally be accessed by and output from only one of these jobs. Sounding and surface observations may be processed by both jobs, however. The output sounding file and the output surface data file from one job would then be concatenated with the corresponding output files from the other job to produce one single upper air and one single surface data file for input to ISAN.

The two DATAPREP jobs are located in the /dprep/ncar and /dprep/generic subdirectories (see Chapter 2). A Makefile exists in each subdirectory to direct the compilation of the respective code and generation of an executable (see Chapter 4). Because code in the /dprep/generic directory always requires user modification before compilation, this code is not pre-compiled in the standard directories but is intended to be copied to a user working directory and compiled there.

For consistency, the same procedure is suggested for the DATAPREP job in the /dprep/ncar subdirectory. Each DATAPREP job reads namelist data from a file named PREP_IN. These data specify a number of job control options and are described later in this Section.

Section 6.2.1 - DATAPREP Output File Structure

ISAN requires at a minimum either gridded pressure level data or sounding data to function. The other of these is optional, and surface observations are also optional. (The ISAN namelist variables described in the next Section are used to select the appropriate option.) Most commonly, all three data types are used by ISAN, and thus generated by DATAPREP. The data content and format of each of the three types of files is described next.

Pressure Data File

The following code extracted from the /dprep/ncar subdirectory performs the opening, writing and formatting, and closing of the pressure data file:

```
      open(10,file=locfn,status='unknown',form='formatted')
      rewind 10

      WRITE(IUN,900) IYEAR,IMONTH,IDATE,IHOUR,NPRZ,NPRX,NPRY
     +     ,WPLON,SPLAT,SPCNPRX,SPCNPRY,(LEVPR(L),L=1,NPRZ)
 900  FORMAT(7I4,4F8.2,/,(13I6))

      do lv=1,nprz
         do n=1,n4
            write(iun,950) ((d(i,j,lv,n),i=1,nprx),j=1,npry)
 950        format(8f10.3)
         enddo
      enddo
      do n=1,ns
         write(iun,950) ((ds(i,j,n),i=1,nprx),j=1,npry)
      enddo

      close(10)
```

The open and rewind statements allow rewriting of an existing pressure file in case one of the same name already exists. The first write statement outputs header information to the file. The variables in the write list are, in order, the year, month, date, and hour of the data in universal (UTC) time, the number of grid points spanning the z-, x-, and y-directions of the pressure grid, the longitude (degrees) of the western boundary and the latitude (degrees) of the southern boundary of the pressure grid, the east-west and the north-south grid spacing (in degrees) of the pressure grid, and the integer pressure in millibars of all pressure levels, which are NPRZ in number. All these variables except SPCNPRX and SPCNPRY are read from the PREP_IN namelist file and are further described below. All variables except LEVPR are written to the first line of the file, and the multiple LEVPR values are on the second line.

The second write statement outputs data stored in the "d" array. The fourth index (n) of this array ranges from 1 to 5 denoting five different field variables: east-west velocity component (m s^{-1}), north-south velocity component (m s^{-1}), temperature (K), geopotential height (m), and relative humidity (fraction). All these variables are written in MKS units. A single record is written for each data type on each pressure level (index lv), and runs through all horizontal points (indexes i and j) with 8 data values to a line in f10.3 format. Subsequent records run through field variables most frequently and vertical pressure levels least frequently.

The third write outputs surface data in five records, one for each of five field variables, with each record running through all horizontal points (indexes i and j) with 8 data values to a line in f10.3 format. This surface data is only available from NMC (not ECMWF) gridded pressure level data and is not currently used in ISAN. The file is closed immediately after this write.

Following is a listing of the first 7 lines of a sample pressure level data file output from DATAPREP. The first 2 lines are the header information and the next 5 lines are the beginning of the u-component (westerly) of velocity at the first pressure level.

```
 79   4   9  12  12 145  73      .00  -90.00     2.50     2.50
 1000   850   700   500   400   300   250   200   150   100    70    50
    -.117     -.071     -.024      .023      .070      .116      .162      .208
     .254      .299      .343      .387      .430      .473      .514      .555
     .594      .632      .669      .705      .740      .773      .804      .834
     .863      .890      .914      .938      .959      .980      .997     1.013
    1.028     1.039     1.050     1.058     1.064     1.068     1.071     1.071
```

Sounding Data File

The following code extracted from the /dprep/ncar subdirectory performs the writing and formatting of the sounding data file:

```
      write(iun,'(a10,2i4,3f12.5)') idsta,lp,lz,xlat,xlon,elev

      do k=1,lp
         write(iun,'(2f12.3,f10.2,f10.4)')
     +         pp(k),zp(k),tp(k)-273.16,rp(k)
      enddo
      do k=1,lz
         write(iun,'(f12.3,2f10.2)')
     +         zz(k),fz(k),dz(k)
      enddo
```

This section of code is executed several times, once for each upper air sounding. The output upper air file is opened and rewound prior to this allowing any pre-existing file of the specified name to be overwritten.

The first write statement writes to one line the 6 header variables: station identification, the number of sounding heights at which temperature and humidity values are reported, the number of sounding heights at which wind speed and direction are reported, and the latitude and longitude (degrees) and elevation (m) of the upper air station.

The second write statement writes sounding pressure (Pa), geopotential height (m), temperature (° C), and relative humidity (fraction) for each of the pressure levels. Each level (containing all four data values) is written on a separate line in the file.

The third write statements writes geopotential height (m), wind speed (m s^{-1}), and wind direction (degrees) for each of the wind levels. Each level (containing all three data values) is written on a separate line in the file.

Following is a listing of the first 57 lines of a sample upper air data file output from DATAPREP. The first line contains the header information including the number of levels (34) of thermodynamic data and the number (17) containing wind data. The next 34 lines contain the thermodynamic data and the following 17 the wind data. The next line is the header line for a second upper air station, and the final 4 lines are the beginning of the thermodynamic data for that sounding.

```
YMD        34  17    76.25000  -119.35001    58.00000
 99200.000       58.000     -25.46      .6890
 98900.000       80.000     -25.86      .6559
 92500.000      575.000     -17.26      .5929
 88200.000      928.000     -19.46      .5351
 83000.000     1379.000     -19.26      .3680
 79500.000     1699.000     -17.26      .1551
 71500.000     2488.000     -21.26      .2426
 70000.000     2646.000     -22.26      .3944
 67600.000     2896.000     -23.46      .1239
 62900.000     3429.000     -26.06      .1469
 60600.000     3697.000     -27.66      .2744
 55700.000     4285.000     -30.46      .1512
 50000.000     5060.000     -35.86      .1728
 40000.000     6570.000     -47.06      .1383
 37100.000     7056.000     -50.66      .1685
 30000.000     8430.000     -55.46      .2694
 25000.000     9610.000     -51.46      .0812
 20000.000    11080.000     -45.86      .0160
 17800.000    11849.000     -48.26      .0086
 15000.000    12980.000     -46.06      .0082
 12900.000    13980.000     -46.46      .0080
 10000.000    15670.000     -47.06      .0078
  8740.000    16561.000     -44.66      .0089
  7260.000    17788.000     -49.06      .0082
  7000.000    18030.000     -48.86      .0083
  5000.000    20230.000     -51.66      .0085
  4390.000    21067.000     -50.66      .0075
  3360.000    22790.000     -52.66      .0080
  3000.000    23520.000     -53.66      .0075
  2420.000    24902.000     -52.86      .0079
  2000.000    26130.000     -56.06      .0078
  1480.000    28023.000     -60.26      .0073
  1050.000    30183.000     -59.26      .0078
  1000.000    30490.000     -59.66      .0076
    58.000      15.40    290.00
   575.000      12.80    300.00
  1202.000      12.30    300.00
  2646.000       9.20    290.00
  5060.000      11.80    290.00
  6570.000      10.30    290.00
  8604.000      11.80    255.00
  9610.000      10.80    255.00
 11080.000       9.20    235.00
 12980.000      11.80    235.00
 15670.000      15.90    250.00
 18030.000      14.40    250.00
 19200.000      16.40    275.00
 20230.000      15.90    265.00
 23520.000      28.30    290.00
 27000.000      36.50    290.00
 30490.000      44.30    300.00
YAK        36  54    59.51000  -139.66000    11.00000
100400.000       40.000      -2.46      .5392
100000.000       71.000      -1.46      .4619
 98800.000      168.000       -.66      .3353
 92500.000      692.000      -4.66      .3515
```

Surface Data File

The following code extracted from the /dprep/ncar subdirectory performs the writing and formatting of the surface data file:

```
      write(iun,'(a6,2f9.3,f8.2,2f9.2,2f9.2,f12.3)')
     +        ASTA,XLAT,XLON,ZX,ffx,ddx,tx,tdx,px
```

This section of code is executed several times, once for each surface station. The output surface data file is opened and rewound prior to this allowing any pre-existing file of the specified name to be overwritten.

The write statement writes to one line the 9 variables: station identification, the latitude and longitude (degrees) and elevation (m) of the surface station, wind speed ($\mathrm{m\,s^{-1}}$), wind direction (degrees), temperature (° C), dew point temperature (° C), and pressure (mb).

Following is a listing of the first 10 lines of a sample surface data file output from DATAPREP. Each line contains data for one surface station.

```
PAIM      66.000 -153.700  389.00     1.50   330.00   -24.96   -27.76    -999.000
WIW       51.670 -105.400  526.00     2.50   240.00    -8.96   -10.96    -999.000
DB28      57.100 -177.700    3.00    21.00   210.00     3.14  -999.00    -999.000
WDU       51.080 -114.220 1235.00     4.60   200.00    -1.26   -16.16    -999.000
GAM       63.770 -171.730    8.00    11.80   120.00    -1.16    -2.26    -999.000
WVI       53.620 -112.030  639.00     3.60   190.00    -7.56   -12.16   93440.000
HMS       46.570 -119.600  223.00     3.60   330.00      .94      .44   99700.000
AKN       58.680 -156.680   17.00     4.10   160.00    -2.76    -3.86    -999.000
WLG       67.200 -130.220   63.00     1.50    70.00   -27.06   -29.96  101410.000
HOM       59.630 -151.500   22.00     4.10    30.00    -7.16   -11.66  101630.000
```

The DATAPREP program in dprep/generic must output files of the same formats as output from dprep/ncar, but the inputs are not defined and must be defined by the user. This program is intended to process any data type other than the standard ncar archives. Examples are special soundings, tethersonde data, and NGM gridded data from ZEPHYR.

Section 6.2.2 - DATAPREP Namelist Variables

Each DATAPREP job reads namelist data from a file named PREP_IN. The namelist variables and their functions are described below. Note the syntax of the namelist file in which every value that is specified must be followed by a comma.

IPRESS Flag of value 0(no) or 1(yes) indicating whether to read a pressure level dataset and to output a pressure file.

IRAWI Flag of value 0(no) or 1(yes) indicating whether to read an upper air (sounding) dataset and to output an upper air file.

ISURFACE Flag of value 0(no) or 1(yes) indicating whether to read a surface dataset and to output a surface file.

NATIMES Specifies number of analysis times to process in this run. The maximum value this can be set to is the value of MAXTIMES in the file **prep_com.h**.

IAHOUR is a multi-valued parameter which specifies the time in hours UTC of each data analysis to be performed. The number of values defined for IAHOUR should equal the value specified for NATIMES.

IADATE is like IAHOUR but specifies the day of the month of each analysis time. The number of values defined for IADATE should equal the value specified for NATIMES.

IAMONTH is like IAHOUR but specifies the month of each analysis time. The number of values defined for IAMONTH should equal the value specified for NATIMES.

IAYEAR is like IAHOUR but specifies the year of each analysis time. The number of values defined for IAYEAR should equal the value specified for NATIMES.

IAPR Input pressure level data file name if ipress = 1. for the pressure stage. If you are doing more than one data time, you only need to specify a file name for the time when it changes. For example, if you had set NATIMES to 5, and the first 3 times used file name "Y01234" and the last two times used file "Y54321", then the data specification for IAPR could be : IAPR(1)='Y01234', IAPR(4)='Y54321',

IARAWI Input archived rawinsonde file name if irawi = 1. It follows the same convention for different times as IAPR.

IASRFCE Input archived surface observation file name if isurface = 1. It follows the same convention for different times as IAPR.

IFNPRESS Output file prefix for the pressure output file. The actual file name will be constructed in the following manner: First, the prefix will be used. Second, an encoded date/time will be appended in the form "dd-mm-yy-hh".

IFNRAWI Output file prefix for the upper air data output file. Actual file names follow the convention mentioned in ifnpress.

IFNSFC Output file prefix for the surface data output file. Actual file names follow the convention mentioned in ifnpress.

NPRX Number of grid points in the east-west direction of the pressure grid. The pressure grid is a subset of the latitude-longitude grid on which the pressure level data input to DATAPREP are defined. It has the same horizontal grid interval and same vertical (pressure) levels, but can cover a reduced horizontal area. Input pressure level data are copied directly to the pressure grid and further processed from that grid which is of manageable size. The pressure grid defined in ISAN must be large enough to have at least two grid points outside the largest RAMS model polar stereographic grid all around the pressure grid perimeter, and the pressure grid defined in DATAPREP and output must be at least as large as that defined in ISAN.

NPRY is like NPRX but pertains to the north-south direction.

NPRZ is like NPRZ but pertains to the vertical direction. In other words, NPRZ is the number of pressure levels on which data are defined in the input gridded pressure level analysis.

PDATA Flag of either 'ECMWF' or 'NMC' to indicate which pressure level data to access. If you are accessing a pressure dataset not from the NCAR archives, new access code will need to be written (see Section 6.2).

WPLON Longitude of western edge of pressure grid to be output (see NPRX above). Allowable range is -180. to 180. degrees.

SPLAT Latitude of southern edge of pressure grid to be output (see NPRX above). Allowable range is -90. to 90.0 degrees. Currently, domains which cross the poles are not handled properly in the pressure stage.

LEVPR Pressure levels to access from the input dataset. Values are integers in millibars in decreasing order. The number of values defined must equal the value specified for NPRZ.

OBSLAT1 is the latitude of the southern boundary of a selected area in which sounding and surface data are to be accessed and written to their respective output files. Allowable range is -90. to 90. Selecting a subset of the entire global area is a convenient means for reducing output file size when only a small region needs to be processed in ISAN.

OBSLAT2 is like obslat1 (see above) but is the latitude of the northern boundary. Allowable range is -90. to 90.

OBSLON1 is like obslat1 (see above) but is the longitude of the western boundary. Allowable range is -180. to 180.

OBSLON2 is like obslat1 (see above) but is the longitude of the eastern boundary. Allowable range is -180. to 180.

Section 6.3 - Structure, Function, and Execution of ISAN

The ISentropic ANalysis package (ISAN) prepares *initialization files* for RAMS model initialization and time-dependent lateral boundary conditions from up to three types of atmospheric data: a gridded pressure level analysis, upper air soundings, and surface observations. Each data type is input from a file of a specific format as generated by the DATAPREP job(s) (Section 6.2).

ISAN is divided into three functional parts or stages. In a particular application, one or more of these stages may not need to be run. The functions are:

Pressure stage:	Inputs gridded data (horizontal wind components, temperature, and relative humidity) defined on pressure levels in the format output from the DATAPREP job(s). Copies all or part of the pressure level data onto the "pressure grid" which usually (but not always) covers a reduced geographical area but has the same pressure levels and horizontal mesh size as those of the input data.
Isentropic/σ_z stage:	Inputs selected sounding and surface data from files in the format output from the DATAPREP job(s). Interpolates the sounding data and the pressure grid data vertically to both a set of specified isentropic surfaces and the σ_z atmospheric model levels. Performs a Barnes objective analysis on each isentropic and σ_z surface which blends the upper air observations and pressure level data according to user-specified options and interpolates them to the atmospheric model polar stereographic horizontal coordinates with a specified degree of smoothing. Performs a separate Barnes objective analysis on the surface observations and surface station elevations resulting in interpolation and smoothing of the data to atmospheric model polar stereographic horizontal coordinates on a surface defined by the station elevations.
Initialization file stage:	Blends the objectively-analyzed isentropic, σ_z, and surface data according to user-specified parameters, and copies the blended data to the atmospheric model grid. Outputs this result to the initialization files to be input to the atmospheric model.

Each of these stages can optionally produce contour and/or vector plots of the analyzed data, along with skew-T plots of the input rawinsondes.

ISAN is a specific option run from the atmospheric model executable RAMS_3b. The namelist variables that control options in ISAN, like those for the atmospheric model (see Chapter 7), are contained in the RAMSIN file. They are grouped into the namelists **$ISAN_CONTROL**, **$ISAN_PRESSURE**, **$ISAN_ISENTROPIC**, and **$ISAN_GRAPH**. Each namelist begins with one of the above identifiers, and ends with the character string $END. Variables in these namelists are described in the following Sections. Note the syntax of the namelist file RAMSIN. Every value that is specified for a namelist variable must be followed by a comma. Note also that an index of configuration and namelist variables for chapters 6, 7, and 8 is located at the end of chapter 8.

ISAN also requires most variables contained in the **$MODEL_GRIDS**, namelist in RAMSIN. Most importantly, the variable RUNTYPE must be set to the character value 'MAKEVFILE' in order to indicate that ISAN, not the atmospheric model, will be run. Variables that specify grid location, size, and resolution are used to determine the grid to which data are to be interpolated in producing the end product of an ISAN run - the initialization files. Use of the **$MODEL_GRIDS** namelist for both ISAN and atmospheric model runs is convenient and helps insure identical variable settings for both jobs. **$MODEL_GRIDS** variables are described in Section 7.2.

Section 6.3.1 - ISAN Configuration Parameters

The following configuration parameters are defined in the **ascoms.h** file. They function as dimensions for several arrays and are responsible for the maximum values that can be set for some namelist variables during execution. These are mentioned for completeness and would not normally need to be changed except in extreme circumstances. They may be set to fairly large values in a standard model installation as their settings do not affect total memory usage very much; most of the memory is allocated dynamically. Note that an index of configuration and namelist variables for chapters 6, 7, and 8 is located at the end of chapter 8.

MAXPR The maximum number of vertical levels that can be used in the pressure data.

MAXISN The maximum number of vertical levels that can be used in the isentropic analysis.

MAXX The maximum number of grid points (in the "x" or east-west direction) in any of the RAMS or pressure grids.

MAXY The maximum number of grid points (in the "y" or north-south direction) in any of the RAMS or pressure grids.

MAXTIMES The maximum number of data analysis times that can be processed in a single run.

MAXAGRDS The maximum number of RAMS grids that can have initialization files generated.

MAXSIGZ The maximum number of vertical levels that can be used in the σ_z analysis.

MAXLEV The maximum number of levels in an input rawinsonde.

Section 6.3.2 - The $ISAN_CONTROL Namelist

MSTAGE is a set of three switches, each with a value of 0 (no) or 1 (yes), indicating whether the pressure, isentropic/σ_z, or initialization stage is to be run. All stages can be run at once or each can be executed individually. The pressure and isentropic stages each output one or more files which subsequent stages will input if run separately.

NATIMES Specifies number of analysis times to process in this run. The maximum value this can be set to is the value of MAXTIMES (see Section 6.4).

IAHOUR is a multi-valued parameter which specifies the time in hours UTC of each data analysis to be performed. The number of values defined for IAHOUR should equal the value specified for NATIMES.

IADATE is like IAHOUR but specifies the day of the month of each analysis time. The number of values defined for IADATE should equal the value specified for NATIMES.

IAMONTH is like IAHOUR but specifies the month of each analysis time. The number of values defined for IAMONTH should equal the value specified for NATIMES.

IAYEAR is like IAHOUR but specifies the year of each analysis time. The number of values defined for IAYEAR should equal the value specified for NATIMES.

IAPR Input pressure level data file names (from Dataprep) for the pressure stage. A separate file name must be specified for each time of NATIMES.

INRAWI Input rawinsonde file name (from Dataprep) for the isentropic/σ_z stage. It follows the same convention for different times as IAPR.

INSRFCE Input surface observation file name (from Dataprep) for the isentropic/σ_z stage. It follows the same convention for different times as IAPR.

IFNPRS Output file prefix for the pressure output file. The actual file name will be constructed in the following manner: First, the prefix will be used. Second, an encoded date/time will be appended in the form "dd-mm-yy-hh". Last, for the isentropic, σ_z, and initialization files, a ".gx" (where "x" is the RAMS grid number) will be added.

IOFLGP Disposition flag for pressure stage output file: 1 - Write file and save, 0 - Do not write file

IFNISN Output file prefix for the isentropic output file. Actual file names follow the convention mentioned in IFNPRS.

IOFLGI Disposition flag for isentropic output file: 1 - Write file and save, 0 - Do not write file

IFNSIG Output file prefix for the σ_z output file. Actual file names follow the convention mentioned in IFNPRS.

IOFLGS Disposition flag for σ_z output file: 1 - Write file and save, 0 - Do not write file

IFNVAR Output file prefix for the initialization (varfile) output file. Actual file names follow the convention mentioned in IFNPRS.

IOFLGV Disposition flag for initialization output file: 1 - Write file and save, 0 - Do not write file

Section 6.3.3 - Pressure Stage: The $ISAN_PRESSURE Namelist

NPRX Number of grid points in the east-west direction of the pressure grid. The pressure grid is a subset of the latitude-longitude grid on which the pressure level data input to ISAN are defined. It has the same horizontal grid interval and same vertical (pressure) levels, but normally covers a reduced horizontal area. Input pressure level data are copied directly to the pressure grid and further processed from that grid which is of manageable size. The pressure grid must be large enough to have at least two grid points outside the largest RAMS model polar stereographic grid all around the pressure grid perimeter.

NPRY is like NPRX but pertains to the north-south direction.

NPRZ is like NPRZ but pertains to the vertical direction. In other words, NPRZ is the number of pressure levels on which data are defined in the input gridded pressure level analysis.

WPLON Longitude of western edge of pressure grid to be output (see NPRX above). Allowable range is -180. to 180. degrees.

SPLAT Latitude of southern edge of pressure grid to be output (see NPRX above). Allowable range is -90. to 90.0 degrees. Currently, domains which cross the poles are not handled properly in the pressure stage.

SPCNPRX East-west grid spacing of pressure data and pressure grid (degrees).

SPCNPRY North-south spacing of pressure data and pressure grid (degrees).

LEVPR Pressure levels to access from the input dataset. Values are integers in millibars in decreasing order. The number of values defined must equal the value specified for NPRZ.

Section 6.3.4 - Isentropic/σ_z Stage: The $ISAN_ISENTROPIC Namelist

NISN Number of isentropic levels on which to perform objective analysis. Selection of this number is a compromise between accuracy (more levels provide greater vertical resolution) and efficiency (fewer levels consume less time). Values for NISN in the 30 to 40 range usually provide sufficient accuracy and are reasonably efficient. Since the optimal isentropic resolution is normally variable and changes with season and location, there are no longer any more automatic ways to set the spacing.

LEVTH Isentropic levels in integer degrees Kelvin of the isentropic grid on which objective analysis is performed. The spacing which normally provides sufficient vertical resolution is 1-2 K near the ground, 3-5 K in the remainder of the troposphere, and 20-30 K in the stratosphere. However, if the hybrid analysis with the σ_z data is used, the levels near the ground may be skipped.

NIGRIDS Number of RAMS grids to analyze. This must be less than or equal to NGRIDS in the **$MODEL_GRIDS** namelist. Although the ability exists to analyze atmospheric data and generate an initialization file independently for each RAMS grid, this option does not yet produce analyses that match smoothly across nested grid boundaries. Thus initialization files for nested grids should be used only for model initialization and not 4DDA (Newtonian nudging during model runtime) (see description for TNUDCENT in Section 7.3). Gradients of prognostic variables initialized at grid boundaries will tend to weaken with time if they are not continuously restored by nudging.

TOPSIGZ The σ_z analysis will be done to all RAMS model levels under this height (in meters).

HYBBOT The hybrid analysis between the isentropic and σ_z datasets is accomplished in this version of ISAN by a weighted blending of the data over a specified, terrain-following layer using a linear (in height above the surface) weighting profile. At the bottom of the layer, the data is completely from the σ_z analysis; at the top of the layer, the data is completely from the isentropic analysis. HYBBOT is the approximate height above the surface of the bottom of this layer. The actual bottom is found as the closest model level to HYBBOT.

HYBTOP is the approximate height above the surface of the top of the isentropic/σ_z blending layer (see HYBBOT above). The actual height of the top is found as the closest model level to HYBTOP.

SFCINF In addition to the blending of the isentropic and σ_z analysis in the creation of the initialization file, the surface data analysis is also blended in a layer near the ground. This only occurs if there are actual surface observations close enough to a grid point and the model level is within a certain vertical distance from the actual station height. This distance is SFCINF (meters). The actual weight given to the surface analysis varies from a full weight if the difference between the model level and station height is 0. to no weight for the surface (full weight for the upper air data) when the difference reaches SFCINF.

SIGZWT An optional weight given to the σ_z analysis in the blending process. If SIGZWT is set to 0., then the initialization file will only contain information from the isentropic analysis. If SIGZWT is set to 1., then the initialization file will contain full information from the σ_z analysis under HYBBOT. Any value between 0. and 1. may be used. For most applications, this should be set to 1.

NFEEDVAR In the creation of the initialization files when more than one grid is analyzed, the data can optionally go through the nesting "feed back" process. The feed back must be done if the 4-dimensional data assimilation options are to be used. Set NFEEDVAR to 1 to do the feed back. Set NFEEDVAR to 0 usually only if you are doing a data analysis on different grid resolutions, will not be running the model, and you want to see the unmodified results of the different resolution analyses. Normally, leave this set to 1.

MAXSTA Maximum number of rawinsondes expected in the data access. Memory will be allocated using this number, so don't set it too big.

MAXSFC Maximum number of surface observations expected in the data access. Memory will be allocated using this number, so don't set it too big.

NONLYS Sometimes just one or a few stations are accessed, for instance, to print a sounding. In cases where not all stations are to be accessed, this is the number of stations to access. If NONLYS is set to 0, then all stations found in the domain are accessed. Since the surface station information is not very long, all of these are printed by default. Therefore, NONLYS only refers to the rawinsondes.

IDONLYS is a character array of station IDs to access in the case where NONLYS (see above) is not set to zero. The number of entries defined for IDONLYS should equal the value specified for NONLYS.

NOTSTA is the number of stations in the domain area to be excluded if they are found. The specific stations to be excluded are indicated by NOTID (see below). Excluding stations in this way is the means for eliminating stations with bad data.

NOTID A character array of the station IDs to be excluded from any further processing (see NOTSTA above). Prefix the station ID with an 'r' for rawinsonde or an 's' for surface observation.

STASEP The surface observations will be objectively analyzed with the Barnes scheme which assumes a uniform data coverage. Frequently, there are 2 surface stations reporting in the same city. As the surface stations are processed, any station within a distance STASEP (in degrees of latitude) of a previously processed station will be discarded unless the new one has less missing data than the previous one, in which case the other previous station will be discarded.

ISTAPLT Flag of value 0(no) or 1(yes) indicating whether to call the graphical routines to plot skew-T's of the rawinsondes. If this is set to 1, the code will only plot the rawinsondes and surface observations and will not do any objective analysis.

ISTAREP Flag of value 0(no) or 1(yes) indicating whether to print rawinsonde values. If this is set to 1, the code will only access and print the observations and will not do any objective analysis.

IUPPER Flag of value 0(no) or 1(yes) indicating whether to do upper air analysis. This is set to 0 when only the surface analysis is required.

IGRIDFL Flag which controls blending of gridded pressure level data and observations. If IGRIDFL = 0, no pressure grid point data is used, only observations. If IGRIDFL = 1, all observations (those not discarded as specified by the STASEP and NOTSTA parameters - see above) and all pressure grid point data are used. If IGRIDFL = 2, all observations (those not discarded) are used, but pressure data are used only from geographic locations that (1) have no sounding closer than the distance specified in GOBRAD and (2) do not have soundings in 3 or 4 quadrants (defined by intersecting east-west and north-south lines) all closer than the distance specified by GOBSEP (see descriptions for GOBRAD and GOBSET below). If IGRIDFL = 3, only gridded pressure level data are used, no observations. If IGRIDFL = 4, the gridded pressure data is taken as a first guess field. The observations are objectively analyzed and applied to this first guess field as deviations. This option is intended as a quick look at the observations and not for generation of the RAMS initial fields.

GRIDWT The implementation of the Barnes scheme in RAMS allows differential weighting of any station or pressure data point to give it more or less influence compared to other stations. GRIDWT has been implemented to reduce the weight of the gridded pressure data set relative to the actual observations. For example, if a rawinsonde observation were made at the same location as a gridded pressure point, a setting of GRIDWT=.1, would give the rawinsonde 10 times more weight than the pressure data. However, remember that the gridded data now are interpolated to the actual RAMS grids, so that there are many more pressure grid points than soundings. Also, there is a pressure grid point at the exact location of the grid to be analyzed. Therefore, a much lower setting of GRIDWT will usually be optimal. The actual value will take some experimentation but a value in the .1 to .001 may be appropriate. Note also that the value is grid dependent and must have as many values specified as NIGRIDS. In general, GRIDWT will become larger as the grid spacing of the RAMS grid increases.

GOBSEP If IGRIDFL (described above) is set to 2, a pressure data grid point will not be used in the objective analysis if it is within GOBSEP degrees of an upper air observation.

GOBRAD If IGRIDFL (described above) is set to 2, a pressure data grid point will not be used in the objective analysis if there are observations in three quadrants of a circle of GOBRAD radius (degrees). The circle is divided into quadrants by north-south and east-west diameters.

WVLNTH In the Barnes objective analysis scheme, a degree of smoothing to be applied is determined from two parameters. The first, WVLNTH, is the wavelength of the data on the isentropic and σ_z surfaces (upper air) specified in kilometers to be retained. The second, RESPON (see below) is the fractional amplitude at which to retain that wavelength. Responses of other wavelengths are determined given a value of .3 set for the "gamma" parameter in the Barnes scheme. Note that WVLNTH is grid dependent and there must be as many values specified as NIGRIDS.

SWVLNTH is like WVLNTH (described above) but pertains to surface rather than upper air data and analysis. It uses the same RESPON value as WVLNTH. Note that SVLNTH is grid dependent and there must be as many values specified as NIGRIDS. for the surface objective analysis. Note that it is grid dependent and there must be as many values specified as NIGRIDS.

RESPON Percentage of specified wavelength amplitudes to be retained in both the upper air and surface analyses (see WVLNTH and SWVLNTH above). Note that RESPON is grid dependent and there must be as many values specified as NIGRIDS.

Section 6.3.5 - Graphical Stage: The $ISAN_GRAPH Namelist

IPLTPRS Main switch for plotting pressure data. 0 - Don't plot anything from the pressure stage. 1 - Plot things from the pressure stage.

IPLTISN Main switch for plotting isentropic data. 0 - Don't plot anything from the isentropic stage. 1 - Plot things from the isentropic stage.

IPLTSIG Main switch for plotting σ_z data. 0 - Don't plot σ_z data. 1 - Plot σ_z data.

IPLTSTA Main switch for plotting station values. These are horizontal plots with values of variables placed at the station locations. Only the surface and isentropic level values may be plotted at this time. 0 - Don't plot station data. 1 - Plot station data.

Variables controlling pressure stage plots:

Graphics that are available from the pressure stage are horizontal plots of various parameters on pressure surfaces.

ILFT1I Grid point value that will be the boundary window on the western (left) side of the plots. If any of these window values are set to 0, the window defaults to the entire domain.

IRGT1I Grid point value that will be the boundary window on the eastern (right) side of the plots. If any of these window values are set to 0, the window defaults to the entire domain.

IBOT1J Grid point value that will be the boundary window on the southern (bottom) side of the plots. If any of these window values are set to 0, the window defaults to the entire domain.

ITOP1J Grid point value that will be the boundary window on the northern (top) side of the plots. If any of these window values are set to 0, the window defaults to the entire domain.

NPLEV Number of pressure levels to plot.

IPLEV Integer array of the pressure levels to be plotted. The number of values specified should equal the value assigned to NPLEV and the pressure level values specified must be also specified for LEVPR (described above).

NFLDU1 Number of fields to be plotted on each pressure level. Each field will be on a separate frame. The total number of frames for the pressure plots is then NPLEV*NFLDU1.

IFLDU1 Character array of field names to plot. The number of character strings specified must equal the value assigned for NFLDU1 (described above). The field names are listed at the end of this chapter.

CONU1 Real array of contour intervals for each field. If set to 0., a default value will be chosen by CONREC. The number of values specified must equal the value assigned for NFLDU1 (described above).

IVELU1 Integer array of velocity vector flags specifying if vectors should be overlayed on the contoured fields. The number of values specified must equal the value assigned for NFLDU1 (described above). 0 - No vectors should be drawn. 1 - Vectors should be drawn at every grid point. 2 - Vectors should be drawn at every other grid point. 3 - Vectors should be drawn at every third grid point.

Variables controlling isentropic stage plots:

Graphics that are available from the isentropic stage are horizontal contour plots of various parameters on isentropic surfaces, horizontal surface contour plots, values plotted on isentropes and the surface, and vertical cross-sections.

ILFT3I Grid point value that will be the boundary window on the western (left) side of the plots. If any of these window values are set to 0, the window defaults to the entire domain.

IRGT3I Grid point value that will be the boundary window on the eastern (right) side of the plots. If any of these window values are set to 0, the window defaults to the entire domain.

IBOT3J Grid point value that will be the boundary window on the southern (bottom) side of the plots. If any of these window values are set to 0, the window defaults to the entire domain.

ITOP3J Grid point value that will be the boundary window on the northern (top) side of the plots. If any of these window values are set to 0, the window defaults to the entire domain.

IUP3BEG Lowest isentropic level to plot.

IUP3END Uppermost isentropic level to plot.

IUP3INC Increment of isentropic levels to plot.

NFLDU3 Number of fields to be plotted on each isentropic level. Each field will be on a separate frame.

IFLDU3 Character array of field names to plot. The number of character strings specified must equal the value assigned for NFLDU3 (described above). The field names are listed at the end of this chapter.

CONU3 Real array of contour intervals for each field. If set to 0., a default value will be chosen by CONREC. The number of values specified must equal the value assigned for NFLDU3 (described above).

IVELU3 Integer array of velocity vector flags specifying if vectors should be overlayed on the contoured fields. The number of values specified must equal the value assigned for NFLDU3 (described above). 0 - No vectors should be drawn. 1 - Vectors should be drawn at every grid point. 2 - Vectors should be drawn at every other grid point. 3 - Vectors should be drawn at every third grid point.

Variables controlling surface plots:

The surface plots use the windowing information from the isentropic plots (ILFT3I, IRGT3I, IBOT3J, ITOP3J).

NFLDS3 Number of fields to be plotted on the surface. Each field will be on a separate frame.

IFLDS3 Character array of field names to plot. The number of character strings specified must equal the value assigned for NFLDS3 (described above). The field names are listed at the end of this chapter.

CONS3 Real array of contour intervals for each field. If set to 0., a default value will be chosen by CONREC. The number of values specified must equal the value assigned for NFLDS3 (described above).

IVELS3 Integer array of velocity vector flags specifying if vectors should be overlayed on the contoured fields. The number of values specified must equal the value assigned for NFLDS3 (described above). 0 - No vectors should be drawn. 1 - Vectors should be drawn at every grid point. 2 - Vectors should be drawn at every other grid point. 3 - Vectors should be drawn at every third grid point.

Variables controlling σ_z plots:

The plots use the windowing information from the isentropic plots (ILFT3I, IRGT3I, IBOT3J, ITOP3J).

ISZBEG Integer index of the lowest σ_z level to plot. These levels pertain to RAMS grid levels.

ISZEND Integer index of the uppermost σ_z level to plot.

ISZINC Increment of integer indexes of σ_z levels to plot.

NFLDSZ Number of fields to be plotted on each σ_z level. Each field will be on a separate frame.

IFLDSZ Character array of field names to plot. The number of character strings specified must equal the value assigned for NFLDSZ (described above). The field names are listed at the end of this chapter.

CONSZ Real array of contour intervals for each field. If set to 0., a default value will be chosen by CONREC. The number of values specified must equal the value assigned for NFLDSZ (described above).

IVELSZ Integer array of velocity vector flags specifying if vectors should be overlayed on the contoured fields. The number of values specified must equal the value assigned for NFLDSZ (described above). 0 - No vectors should be drawn. 1 - Vectors should be drawn at every grid point. 2 - Vectors should be drawn at every other grid point. 3 - Vectors should be drawn at every third grid point.

Variables controlling station plots: The surface plots use the windowing information from the isentropic plots (ILFT3I, IRGT3I, IBOT3J, ITOP3J).

NPLTRAW Number of rawinsonde plots per page. As each rawinsonde is accessed, a two panel graph is made of the basic variables (u, v, potential temperature, relative humidity). The purpose of these plots is to check for bad data in the report. NPLTRAW is a target numer of how many rawinsondes to graph on each frame. For instance, a value of 16 will plot a 4 by 4 array of 16 rawinsondes.

NSTIS3 Number of fields to be plotted from the surface data. Each field will be on a separate frame.

ISTIS3 Character array of field names to plot. The number of values specified must equal the value assigned for NSTIS3 (described above). The field names are listed at the end of this chapter.

Variables controlling vertical cross-sections:

NCROSS3 Number of cross-sections to plot.

The following 4 variables must have as many values specified as the value assigned to NCROSS3:

ICRTYP3 Integer array containing cross-section type: 1 - East-west cross-section. 2 - North-south cross-section.

ICRA3 Integer array of grid point values of the left boundary.

ICRB3 Integer array of grid point values of the right boundary.

ICRL3 Integer array of grid point values of the cross-section locations.

NCRFLD3 Integer array containing the number of fields to plot on each cross-section. There will be NCROSS3*NCRFLD3 frames each with isentropes and another field contoured.

ICRFLD3 Character array of field names to plot. The field names are listed at the end of this chapter. The number of character strings specified must equal the value assigned to NCRFLD3 (described above).

THCON3 Real array of the contour increment of the isentropes. The number of values specified must equal the value assigned to NCRFLD3 (described above).

ACON3 Real array of the contour increment of the other field. The number of values specified must equal the value assigned to NCRFLD3 (described above).

Field names for the graphical stages

Pressure plots:

U East-west wind component

V North-south wind component

TEMP Temperature

GEO Geopotential (height)

RELHUM Relative humidity

MIXRAT Vapor mixing ratio

THETA Potential temperature

SPEED Wind speed

ENERGY Moist static energy

THETAE Equivalent potential temperature

SPRESS Sea-level pressure

Isentropic plots:

U East-west wind component

V North-south wind component

PRESS Pressure

MONTSTR Montgomery streamfunction

RELHUM Relative humidity

THETA Potential temperature

SPRESS Sea-level pressure

Surface station plots:

U	East-west wind component
V	North-south wind component
PRESS	Pressure
RELHUM	Relative humidity
MIXRAT	Vapor mixing ratio
TEMP	Temperature

<u>σ_z **plots:**</u>

U	East-west wind component
V	North-south wind component
PRESS	Pressure
THETA	Potential temperature
RELHUM	Relative humidity
MIXRAT	Vapor mixing ratio
TEMP	Temperature
SPEED	Wind speed
ENERGY	Moist static energy
THETAE	Equivalent potential temperature

CHAPTER 7 - THE RAMS ATMOSPHERIC MODEL

Specifying values for each of the variables in the atmospheric model namelists is the principal way that a user sets up the desired model configuration and selects the many options available for the particular model run to be done. The atmospheric model component of RAMS contains 5 namelists, which have the names **$MODEL_GRIDS**, **$MODEL_FILE_INFO**, **$MODEL_OPTIONS**, **$MODEL_SOUND**, and **$MODEL_PRINT**. These namelists are all contained in the data file called **RAMSIN** in the **model** directory. Each namelist in this file begins with one of the above identifiers, and ends with the character string $END. All variables in these namelists have sample values assigned to them in the **RAMSIN** file, but the user will need to edit many of them for a specific simulation or forecast. In a separate file, **rconfig.h**, in the **include** directory, the user sets values of 7 different parameters used in dimensioning the common blocks of the atmospheric model.

Following are descriptions of all atmospheric model parameters and namelist variables, detailing what they represent or control, what acceptable settings for them are, and how they are interrelated. In addition, descriptions of many of the namelist variables include recommended settings based on our experience and philosophy. Sometimes, additional information on a namelist variable can be found in descriptions of related variables. For this reason, a complete read-through of all namelist variables is recommended. *Note as a matter of syntax that each value assigned to any namelist variable in the* **RAMSIN** *file must be followed by a comma.* An index of configuration and namelist variables for chapters 6, 7, and 8 is located at the end of chapter 8.

Section 7.1 - The rconfig.h Parameters

The 7 parameters defined subsequently in the **rconfig.h** file are used as dimensions for many of the variables in the atmospheric model global common blocks. For this reason, any time one of these parameters is changed, all files in the **model, isan**, and **van** directories must be recompiled. The parameters are only required for setting limits on common block array space. Often, the model is run well within those limits. The parameters should be set large enough for the required model space, and for convenience, large enough to fit any anticipated expansions of required model space, but not unreasonably large such that excessive computer memory is wasted. A suggestion is to set MAXGRDS=10, NXPMAX=200, NYPMAX=200, NZPMAX=150, NZGMAX to 20, and MAXSCLR to 150, which can accommodate the majority of RAMS configurations used. In the following, the parameters are described individually.

MAXGRDS specifies the maximum number of grids which may be used in a model run. The number of grids actually used in a run is specified by the namelist variable NGRIDS (see in Section 7.2). NGRIDS itself may be changed from one model run to the next using the same executable, as long as it does not exceed the value of MAXGRDS used in compiling the model. Thus, MAXGRDS should be set to the largest number of grids which will be used in a series of runs made from the same compiled code.

NXPMAX specifies the maximum number of grid points in the x-direction to be used on any grid. The actual numbers of grid points spanning the x-direction on individual grids are specified in the multiple values of the namelist variable NNXP (see in Section 7.2); NXPMAX must be equal to or larger than those values.

NYPMAX specifies the maximum number of grid points in the y-direction to be used on any grid. The actual numbers of grid points spanning the y-direction on individual grids are specified in the multiple values of the namelist variable NNYP (see in Section 7.2); NYPMAX must be equal to or larger than those values.

NZPMAX specifies the maximum number of grid points in the z-direction to be used on any grid. The actual numbers of grid points spanning the z-direction on individual grids are specified in the multiple values of the namelist variable NNZP (see in Section 7.2); NZPMAX must be equal to or larger than those values.

NZGMAX specifies the maximum number of vertical levels to be used on any grid in the soil model. The actual numbers of levels on individual grids are specified in the multiple values of the namelist variable NNZG (see in Section 7.2); NZGMAX must be equal to or larger than those values.

MAXSCLR specifies the maximum number of scalars that may be automatically added to a simulation. The actual number of added scalars is specified in namelist parameter NADDSC, described in Section 7.4. MAXSCLR must be equal to or larger than NADDSC. **Caution: Do not set MAXSCLR to a value less than 12, or else another parameter in the include file rcommons.h must also be changed, as per the comment in that file**.

MAXDIM must be set to the largest of the values in NXPMAX, NYPMAX, NZPMAX, and NZGMAX.

Section 7.2 - The $MODEL_GRIDS Namelist

The **$MODEL_GRIDS** namelist provides information to the model primarily on the structure of the one or more nested grids used in a simulation, including location, mesh size, number of mesh points, spatial nesting relationships, time step length, and duration of the run. Most of these variables are arrays dimensioned to the parameter MAXGRDS, which is set in the file **RAMSIN**. Each value in the array applies to a different grid, several of which may be activated if the user wishes to employ grid nesting. In the **$MODEL_GRIDS** and other namelists, the multiple values are separated by commas following the variable name and an equals sign. The **RAMSIN** file must have at least as many values of the grid-dependent variables specified as the value assigned to NGRIDS (see in this Section), but no more than the value of MAXGRDS.

EXPNME is a single-valued character string of up to 64 characters into which the user may put any message (*i.e.*, the experiment name) identifying the particular simulation or run. The character string is written in the standard output file (unit 6) and simply serves as a convenient means of labeling the output for the user's own purposes.

IOTYPE is a single-valued flag where the user specifies whether disk I/O is to be performed in this run. If IOTYPE is set to 0, no I/O is done, while if IOTYPE is set to 1, I/O will be done in such a way that central memory contains a complete set of field variables for only one grid at a time. If IOTYPE is set to -1, the model does not perform a normal run, but simply prints out a diagnostic message stating the amount of central memory required to contain information from (1) all grids and (2) the largest grid given the user-specified configuration. In general, the disk I/O feature (IOTYPE=1) is obsolete and should not be used.

RUNTYPE is a single-valued character variable specifying one of four ways in which the model is to be run. In many situations, a RUNTYPE of 'MAKESFC' should be the first run that is made. This type of run will start the model and create surface characteristic datasets for all grids which will be used during subsequent runs. The characteristics include the topography, land percent, sea surface temperature, and vegetation class, which, depending on the form of the initializing datasets, may take several minutes to compute for each grid. These characteristics will be input and initialized according to the specifications of ITOPTFLG, IPCTLFLG, ISSTFLG, and IVEGTFLG, then new files will be created under the naming convention specified by the variable SFCFILES (see below). These files can then be quickly re-read on subsequent runs of the model or ISAN. This is also an easier way to add a grid with higher resolution topography on a 'HISTORY' start.

A value of 'MAKEVFILE' is the choice which runs ISAN (see Chapter 6) which performs an objective analysis of one or more observational datasets, and produces one or more resultant *initialization files*. The initialization files contain a set of atmospheric fields defined on the model grids configured as specified by other variables in the **$MODEL_GRIDS** namelist, and are to be used in a later run for initializing a simulation, and optionally as 4-D assimilation data during the course of the simulation. The MAKEVFILE option is used only if the model is to be initialized from a complete 3-D objective analysis, (the INITIAL variable in the **$MODEL_GRIDS** namelist set to 2), and is the means by which the observational data are processed and interpolated to the model grid(s). The most common way to use this option is to generate initialization files from the objective analysis package every 12 hours, since that is the time interval at which upper-air data are normally collected. The total number of initialization files required depends on the duration of the simulation to be performed and the particular grids which will be initialized from them (and optionally will use them for 4DDA). The first initialization file must correspond to the starting time of the simulation, while the last initialization file must correspond to a time between zero and 12 hours after the end of the simulation. Parameters which control the input and analysis of 3-D observational data are described in Chapter 6.

If RUNTYPE is set to 'INITIAL', the current model run is designated to be the first of a simulation (see definitions of *run* and *simulation* in the entry for TIMMAX below). This means that a simulation is begun at time zero, and that all atmospheric and soil prognostic variables are initialized either horizontally homogeneously from sounding and soil data in the **$MODEL_SOUND** and **$MODEL_OPTIONS** namelists (INITIAL set to 1 in this case; see in section 7.3), or are read from an initialization file prepared in an earlier ISAN run (INITIAL set to 2).

If RUNTYPE is set to 'HISTORY', the model is to be history restarted, and the atmospheric and soil prognostic variables are read from a history file, which was written by the model on a previous run. This option is used when a simulation is carried out over a series of two or more runs. (See also the entries for IOUTPUT and HFILOUT in Section 7.3.)

TIMEUNIT is a single-valued character variable where the user specifies the time units in which the variables TIMMAX, TIMSTR, and VTIME are expressed. These variables are described in this or the following Section. The allowable values of TIMEUNIT are 's' to denote 'seconds', 'm' to denote 'minutes', and 'h' to denote 'hours'. This option allows numerical values for the above variables to be specified within convenient ranges for any simulation, which may range in duration from seconds to years.

TIMMAX specifies the time during a simulation in units of seconds, minutes, or hours (see description of TIMEUNIT in this Section) when the current run is to stop. We take care here to define the terms *simulation* and *run*. A *simulation* is an entire integration of the atmospheric model from initial conditions to a final time. It consists of one or more *runs*, which are individual submissions of the model code to the computer for execution. The TIMMAX variable represents the total time elapsed from the beginning of an entire simulation, starting with the first run, and is cumulative over successive runs. However, it denotes the time at which the current run is to terminate, rather than when the entire simulation is to end. (See also the entry for TIMSTR in Section 7.3.)

NGRIDS is a single-valued variable which specifies the number of grids to be activated for the model run. If set to 1, only a single grid covering the entire spatial domain of the simulation will be activated. A setting of 2 will activate a nested grid within the first grid. A nested grid is used to attain a higher spatial resolution in a limited area of the simulation domain. This finer-resolution second grid communicates with the coarser-resolution parent grid via two-way interaction following the scheme designed by Clark and Farley. Figure 1 shows an example of a lattice consisting of two small nested grids. The example is a horizontal cross section showing the relative positions of the horizontal velocity components (u,v) and the thermodynamic variables (at locations denoted by t) according to the Arakawa-C grid stagger used in RAMS. Values on the coarse grid are denoted by upper-case letters, while those on the fine grid are indicated in lower-case letters. NGRIDS can be set to as many nested grids as desired. However, the variable MAXGRDS (discussed in Section 7.1) must be set to a value at least as large as NGRIDS. It is recommended that NGRIDS be set to 1 the first time a simulation is begun, in order to test, at the relatively low cost incurred by the single grid, the remainder of the model configuration specified by the other namelist variables and other data inputs for the particular application. Once these tests are completed, standard procedure is to add finer grids when necessary in regions where higher resolution is desired, such as over complex terrain, or when trying to resolve a particular meteorological feature such as a sea-breeze circulation or a thunderstorm. It is common for a given simulation to consist of a series of sequential runs, each new run advancing the simulation forward in time beyond the previous run. RAMS allows the user to increase the value of NGRIDS from one run to the next, thus spawning new fine grids during the course of the simulation. It must be remembered that the variable MAXGRDS, needs to be set to at least to at least the maximum value of NGRIDS which will be used at any time during the simulation.

NNXP specifies the number of grid cells to span the computational domain in the x-direction. It is an array dimensioned to MAXGRDS, and has a value for each grid activated. These values are written in a single line in the namelist and are separated by commas. The first value in the line refers to grid number 1, the second to grid 2, and so on. The number of grid cells spanning each spatial direction (x,y,z) consists of interior cells where normal evaluation of all terms (such as advective tendencies) in the governing equations is performed, plus a boundary point at each end where variables are computed only by boundary conditions based on interior values. In the example of Figure 1, NXP (which is NNXP for grid 1) is set to 8, and the first and last grid cells in the x-direction (index values I=1 and I=8) are the boundary points. Thus, the actual computational domain may be considered to comprise only the interior cells, those shown bounded by lattice lines in the figure, and the user may wish to set NNXP to a value 2 larger than the intended domain size. For the U velocity points, boundary conditions are applied at I=1 and I=7, with normal evaluation of the governing equations being performed at points in between. The U values at I=8 are dummy points and are never used. The fine grid in Figure 1 also has nxp (which is NNXP for grid 2) equal to 8. (See also the descriptions for NSTRATX and NINEST).

NNYP corresponds to NNXP, but applies to the y-direction. The example in Figure 1 has NYP (NNYP for the coarse grid) equal to 9 and nyp (NNYP for the fine grid) equal to 6. If NNYP is set to 1, the simulation is two-dimensional, and NNYP must then also be set to 1 for all grids activated. A two-dimensional simulation may not be done with NNXP equal to 1 and NNYP greater than 1, *i.e.,* only x-z cross sections can be used for 2-D runs. Pseudo-one-dimensional simulations may be configured by setting NNYP to 1 and NNXP to 5 coupled with cyclic boundary conditions in the x-direction (see description for IBND in the **$MODEL_OPTIONS** namelist). For the example in Figure 1, boundary conditions are applied to T and U at J=1 and J=9, and to V at J=1 and J=8. V at J=9 is a dummy value and is never used.

NNZP corresponds to NNXP, but applies to the z-direction. Figure 2 shows an x-z cross section through the model grid analogous to the x-y cross section of Figure 1. Vertical indices for the coarse and fine grids are K and k, respectively, and NZP and nzp (which are the NNZP values for grids 1 and 2, respectively) are the corresponding upper limits of these indices. Vertical velocity components are W and w. The example consists of 7 interior grid cell levels on the coarse grid (K=2 to K=8 of T cells) with boundary cells K=1 below and K=9 above. The ground surface is located at the K=1 W level, while the model top is considered to be at the K=8 W level. It is at this top level where W is set to 0 if a rigid lid is chosen as the model top boundary condition. W at K=9 is a dummy point and is never used. Note that this example is for illustration only and is **not** realistic for a real model run since *NNZP must be set to a value of 10 or greater for each grid, and must in addition be greater than or equal to NNZG (see in this Section).*

NNZG specifies the number of levels to be activated in the soil model. The soil model consists of a grid representing finite volumes in the uppermost meter or so of the ground. Soil temperature and moisture are prognosed on this grid based on equations governing their internal transport and external exchange with the atmosphere. The horizontal dimensions and resolution of the soil model are identical with those of the atmospheric model grid above, with NNXP and NNYP applying equally to both models. NNZG can be set to around 10 for most applications. NNZG must have the same value for all grids which contact the ground (see description for NNSTBOT in this Section).

NXTNEST identifies the parent grid of each nested grid, *i.e.*, the coarser grid with which the nested grid has direct communication. Defining values in this array is necessary because different options are available for specifying the interrelation between grids. All grids from 1 up to NGRIDS are numbered consecutively with no integers being skipped, and the coarser grid in which a finer grid is nested always has a lower grid number than that finer grid. Hence, grid 1 is always the coarsest grid, covering the entire computational domain, and grid 2 is always nested directly within grid 1. Thus, NXTNEST for grid 2 (the second value for NXTNEST) must always be set to 1. (NXTNEST should also be set to 1 for grid 1.) However, grid 3 can either be nested in grid 1 (by setting NXTNEST for grid 3 equal to 1), meaning that it is at a parallel nesting status with grid 2, or grid 3 can be nested within grid 2 (by setting NXTNEST for grid 3 equal to 2), such that it is two nesting levels removed from grid 1.

IHTRAN is a single-valued flag specifying the type of horizontal grid transformation to be used in a model simulation. If it is set to 0, the model grid will be cartesian, with uniform horizontal spacing used throughout the domain. This option is often used in small computational domains, or where the simulation is of an idealized situation, not corresponding to any particular geographic location. The basic assumption of this grid is that earth curvature effects are negligible. Setting IHTRAN to 1 sets up a polar stereographic coordinate system for the simulation. While the model grid spacing itself appears uniform and cartesian, map factors are activated which cause the earth distances mapped from model distances to be of variable resolution. A polar stereographic projection is a mapping between the spherical earth and a plane (assumed to correspond to zero elevation in the cartesian model grid) tangent to the earth, with all projection lines emanating from the point on the earth's surface opposite or antipodal to the point of tangency. The cartesian model plane is centered around the point of tangency, which may be placed anywhere on the earth (the pole in this case refers to the point of tangency, specified in namelist variables POLELAT and POLELON (see in this Section), not to either geographic pole). Thus, size ratio in the center of the model domain is unity between the model grid and the earth, and slowly increases outward toward the model lateral boundaries. This transformation allows the model to cover geographic domains up to hemispheric in size, and avoids the numerical problems encountered near the geographic poles when latitude-longitude coordinates are used.

DELTAX specifies the horizontal grid spacing or grid cell size in the x-direction of the model coarse grid in units of meters. This very important parameter determines the scale of meteorological features which are resolvable on the grid, and together with the number of grid cells in the x-direction, NNXP (see in this Section), determines the total x-direction span of the model domain. All spacings on the finer nested grids are derived from this value of DELTAX along with the individual grid spacing ratios NSTRATX (see in this Section). In choosing DELTAX for a simulation, the user needs to carefully decide what essential meteorological phenomenon is to be modeled, what the required resolution is for adequately simulating this phenomenon, how large a domain is required to contain the environment of the phenomenon, what the duration of the simulation will be, whether nested grids will be used, and how much computer memory and CPU time are available. Other factors too may influence the choice for DELTAX, for example, whether or not the hydrostatic option is used.

DELTAY is similar to DELTAX, but specifies the y-coordinate, rather than x-coordinate dimension of a grid cell on the model coarse grid. One should normally set DELTAX and DELTAY to the same value, since different values may numerically force horizontal anisotropy in the simulated fields. DELTAY should still be set equal to DELTAX even in a 2D run.

DELTAZ is similar to DELTAX, but specifies the vertical, rather than x-coordinate, dimension of a grid cell on the model coarse grid. To be more precise, DELTAZ is the vertical grid spacing of the lowest grid level in the domain, and is equal to the vertical grid spacing at higher levels only if constant grid spacing is used in the vertical. (See also descriptions for DZRAT and DZMAX.) However, if DELTAZ is set to 0., it serves as a flag to instruct the model to read the ZZ variable from the namelist (see in this Section) for obtaining vertical grid levels directly. As for the case of DELTAX, selecting an optimal value for DELTAZ involves a number of considerations. An additional consideration in this case is that a small value for DELTAZ in a domain containing steeply-sloping topography requires high horizontal resolution for computational stability. A rule of thumb here is that the terrain height difference between adjacent grid cells should not exceed about 3 to 5 times DELTAZ.

DZRAT is used in conjunction with DELTAZ (see in this Section), and specifies the ratio in vertical grid cell dimension between adjacent levels on the coarse grid. DZRAT is used as a convenient means for vertically stretching the grid, which is a very common practice to obtain high vertical resolution near the ground and lower resolution at higher levels. This method establishes a geometrically-stretched grid, wherein the expansion ratio is constant between consecutive levels. Geometric stretching minimizes the maximum expansion ratio between consecutive levels for a given net stretching over a given number of vertical levels. It is desirable to keep the ratio between consecutive levels small, as large ratios destroy the second-order accuracy of the vertical differencing in the model. Values for DZRAT of 1.1 or even 1.2 are considered acceptable. To illustrate with an example, if DELTAZ is set to 100 and DZRAT is set to 1.1, the first (lowest) grid level will be 100 m thick, the next 1.1 times the first or 110 m, the third 1.1 times the second or 121 m, etc.

DZMAX specifies an upper bound on vertical grid spacing anywhere in the model coarse grid, and is used in conjunction with DZRAT (see in this Section) to prevent the automatic geometric grid stretching from causing overly large vertical grid cell sizes in the upper levels of the model. For example, if DZRAT is set to 1.1, the vertical grid spacing will increase by an order of magnitude over 25 levels. This means that if the first level above the surface is 200 m thick, the 26th level will be over 2 km thick. By specifying DZMAX to a number such as 1000, the geometric expansion will halt when the 1000 m vertical thickness is reached, and all levels above will be made 1000 m thick.

ZZ is an array of heights of coarse grid levels in meters, beginning at the ground (ZZ = 0) and continuing to the top level in the model. The number of values specified for ZZ must equal NNZP for grid 1, which is specified in the $MODEL_GRIDS namelist. Specifying ZZ directly instead of using DELTAZ, DZRAT, and DZMAX (see in this Section) is an alternative means of setting the vertical grid structure, and of course offers more flexibility because ZZ can be specified in many ways other than constant or geometrically expanded spacing. However, when using direct specification via ZZ, care needs to be taken not to cause two consecutive levels to have grossly different vertical thicknesses, such as a factor of 2, as this corrupts the accuracy of the model's vertical differencing schemes. For the model to use the specified ZZ values, DELTAZ must be set to 0.

DTLONG specifies the length of the timestep in seconds at which most processes on the coarse grid will be updated. This is the primary timestep in the model to which all others relate. For example, the values set for NNDTRAT (see in this Section), set integer ratios of timesteps between the coarse and fine grids, such that an appropriate fraction of DTLONG is the actual timestep used on a fine grid. One should set DTLONG reasonably close to, but below the limiting value for computational stability of the model. Given the full model equation set, this value is determined by a number of factors, including wind velocity, internal gravity wave speed, external gravity wave speed (related in part to the height of the model domain), vertical and horizontal grid spacing, and maximum terrain slope. It is not easy to provide a comprehensive formula for estimating an optimal value for DTLONG, and the user should generally try a few values when beginning a new simulation to find a suitable value. However, some guidelines can be provided based on experience. For a deep model domain, say 20 km or more, the model timestep can usually be 5 or 6 seconds for a 1 km horizontal resolution, 30 seconds for a 10 km horizontal resolution, and 90 seconds (for the nonhydrostatic option) for a horizontal resolution around 50 km. We have found that if the hybrid time differencing scheme is used (see ITMDIFF in Section 7.4), time steps can reach 2 or 3 minutes or even longer for coarse grids of 100 km or more. The leapfrog time differencing scheme appears to require a more restrictive timestep. The hydrostatic model can run with a 120 second timestep or longer for coarse resolutions in the 100 km range. In a shallow domain 3 km deep, we have been able to set DTLONG to 6 seconds with a 125 meter horizontal spacing. When steep terrain is used (the relevant measure of steepness appears to be the maximum change in terrain height between two adjacent grid cells in vertical grid spacing units, with a value over about 5 being considered steep), DTLONG is often required to be smaller.

NRATIO specifies the number of *short* or *acoustic* timesteps performed for each *long* timestep on a grid, and has the same value for all grids. The *short* timesteps are used for evaluating the pressure gradient force and divergence equation terms and applying them to the velocity and pressure fields, *i.e.*, the terms involving the propagation of sound waves, including Lamb waves. The *long* timestep is used to evaluate most other terms, including advection, diffusion, and Coriolis force. This practice of *time splitting* of the prognostic equations allows full explicit computation (on the short timestep) of the terms governing the rapidly moving sound waves, while the terms governing slower processes can be performed on the long timestep. NRATIO has nothing to do with the relation between timesteps of one grid and another. On the coarse grid, the long timestep is defined by DTLONG, defined above. A separate variable DTLT, which is not defined in the namelists but is internal to the code, is the specific timestep length for each individual grid. It is equal to DTLONG (defined above) on the coarse grid, and is determined for the finer grids from a combination of DTLONG and NNDTRAT, defined below in this Section. The length of the short (sound wave) timestep is twice the ratio DTLT/NRATIO for leapfrog or hybrid time differencing (see ITMDIFF), and equal to the ratio DTLT/NRATIO for forward time differencing. This short timestep must be short enough (by using a sufficiently large value of NRATIO) that sound cannot travel more than one half the horizontal grid spacing during the short timestep. In the nonhydrostatic model, sound normally travels more than 300 meters per second, but may be artificially slowed down in the model by appropriate setting of the namelist variable SSPCT (see in this Section). NRATIO should usually be set to a value from 2 to 8. In the hydrostatic model, the Lamb and external gravity waves travel near 300 meters per second also. However, NRATIO should not be set larger than 4. In either case, if it seems NRATIO should be set larger, then DTLT should probably be reduced, either by reducing DTLONG or by increasing NNDTRAT.

IMONTH1 specifies the month of the year when the simulation begins. It is used in conjunction with the namelist parameters STRTIM and IDATE1 (see in this Section) to determine the proper solar declination angle for a simulation. It also serves as an identifier of the simulation for the user's information (see also description of EXPNME in this Section).

IDATE1 specifies the date of the month when the simulation begins. It is used in conjunction with the namelist parameters STRTIM and IMONTH1 (see in this Section) to determine the proper solar declination angle for a simulation. It also serves as an identifier of the simulation for the user's information (see also description of EXPNME in this Section).

IYEAR1 specifies the year when the simulation begins. It serves as an identifier of the simulation, along with IMONTH1 and IDATE1 (see in this Section) for the user's information (see also description of EXPNME in this Section).

STRTIM specifies the Coordinated Universal Time (UTC) (or GMT) in hours when a model simulation begins (see definition of *simulation* in the description of TIMMAX in this Section). This parameter is used in the shortwave radiation parameterization (see description for ISWRTYP) to coordinate the daily solar cycle with the simulation, and for labeling UTC times on plots of model output.

NSTRATX specifies the number of grid cells in the x-direction in a fine grid which span the x dimension of a single grid cell of the coarser parent grid to that fine grid. In other words, NSTRATX is the factor of increase in resolution in the x-direction between consecutive nesting levels. For example, if the third value of NSTRATX is set to 4 and the third value of NXTNEST is set to 2, grid 3 will have 4 times the resolution or one fourth the grid cell size of grid 2 in the x-direction. If in the same example, the second value of NSTRATX were also 4, grid 3 would have 16 times the resolution of grid 1 in the x-direction. The example shown in Figure 1 has NSTRATX set to 2 for grid 2. An important constraint between the values of NSTRATX and NNXP must be observed for fine nested grids. A fine grid must exactly cover an integer number of grid cells in its coarser parent grid with its *interior* cells (defined in the entry for NNXP in this Section), and must have the two additional boundary cells. Thus, NNXP for any nested grid must be some integer number (of coarser grid cells covered - 2 in the example of Figure 1) times NSTRATX for that nested grid (3 in the example of Figure 1) plus 2. For NSTRATX and for other related nest configuration variables, if the required combinations of these are not met, the model will stop and issue a message suggesting an acceptable value. Values of 3 and 4 have been most commonly used for NSTRATX. A value of 2 is not usually considered a sufficiently large increase in resolution to make nesting worthwhile, while values of 5 and larger are more prone to cause reflections at the nesting boundaries due to the larger disparity of resolvability between the two grids. (See also the entry for NINEST in this Section.)

NSTRATY corresponds to NSTRATX, but applies to the grid resolution ratio in the y-direction. In the example of Figure 1, NSTRATY is set to 2. If NNYP is set to 1, so that the simulation is 2-D, NSTRATY must be set to 1 for all grids activated.

NNDTRAT specifies the number of timesteps which a fine grid is to be executed to advance it in time the same amount as a single timestep of the coarser parent grid. Because the fine grid has a smaller cell size, it must normally be run with a smaller time step for numerical stability, and the time step length will automatically be made inversely proportional to NNDTRAT. As an example, if the second value of NNDTRAT is set to 4, and the timestep on grid 1 happens to be set to 20 seconds (see the entry for DTLONG in this Section), grid 2 will run on a 5-second timestep, four of which will be performed for each timestep of grid 1. The first value of NNDTRAT, which applies to grid 1 and therefore has no other meaning, is used as a flag: A value of 1 means that all the other values for NNDTRAT (the second, third, and so on) will be used as specified, while a value of 0 means that all the other values will be ignored, and the model will compute its own values directly from the larger of NSTRATX and NSTRATY. This latter practice is often over-restrictive, forcing the timestep on a fine grid to be smaller than necessary. The longest timestep on which a grid can be run stably is not directly proportional to the cell size, but tends to increase more slowly. Thus, a finer grid nested within a coarse grid at say a 3:1 ratio can normally be run at longer than the expected 1/3 of the coarse grid timestep; 1/2 of the coarse grid timestep may be acceptable.

NESTZ is a single-valued parameter whose absolute value denotes the number of the nested grid, if any, which is to be nested with a higher vertical resolution than its parent (See description of NSTRATZ in this section). Such vertical nesting may be done for only one of all model nested grids. Because the vertical nesting ratio can be made variable with height between this nested grid and its parent, the vertical grid spacing of the nested grid may jump suddenly from one vertical level to the next. In order to reduce the relative amount of jump (that is, the ratio of vertical grid spacing between consecutive levels), the model can automatically compensate somewhat by making adjustments to the vertical grid spacings in the parent and coarser grids. This option is activated by giving NESTZ a negative value. If NESTZ is set to 0, no vertical nesting will be done.

NSTRATZ is related to NSTRATX, but applies to the grid resolution ratio in the z-direction. Unlike NSTRATX, however, NSTRATZ is not a grid-dependent parameter, but refers to the vertical nest ratio between a particular nested grid, specified by the user in the namelist variable NESTZ (see in this section), and its parent. NSTRATZ is nevertheless a multi-valued parameter, to allow the vertical nest ratio to be varied with height. Each value specified signifies the number of fine grid levels contained in a single level of the parent grid, beginning with the K=1 level of the parent grid and continuing upward (see Figure 2). Only those levels up to the last one having a vertical nest ratio greater than 1 need be entered in the namelist, because all others are assigned a value of 1 by default. Also, the first value specified in the namelist is not actually used, because it is re-assigned in the code. Nevertheless, some value must be entered to serve as a placeholder in the namelist so that the subsequent values are in the proper location. This feature has most often been used to enhance vertical resolution close to the ground on a nested grid, in which case a decreasing sequence such as NSTRATZ=4,4,4,4,3,3,3,2,2,2, may be used. The feature may also be used to enhance vertical resolution in a selected elevated region, such as a cloud layer. Use of vertical nesting is not necessarily the best means of increasing vertical resolution in all cases. The alternative method of increasing vertical resolution over a limited vertical extent of the model domain (but over the entire horizontal extent) has been to stretch the vertical grid spacing (see the entry for DZRAT in this Section). For this choice, NSTRATZ could be set to 1, although a combination of both methods of stretching is possible, if desired.

POLELAT and **POLELON** specify the geographic latitude and longitude in degrees (ranging from -90 to 90 and -180 to 180, respectively) of the pole point, or point of tangency between the model polar stereographic grid and the earth.

CENTLAT and **CENTLON** are multi-valued, grid-dependent parameters in which the user specifies the geographic latitude and longitude in degrees (ranging from -90 to 90 and -180 to 180, respectively) of the center of each model grid. They are used to define the position of the grids in earth coordinates for the purpose of determining the Coriolis force, the solar radiation, and the location of the model domain relative to observed meteorological and land surface data. For the coarsest model grid, CENTLAT and CENTLON are the only means of specifying the center location. Nested grid locations may also be determined from variables NINEST, NJNEST, NSTRATX, and NSTRATY (see descriptions in this Section), given the locations of their parents and ultimately relating each to the coarse grid location and size. Only if NINEST and NJNEST are given zero values will CENTLAT and CENTLON be used to determine the nested grid location. In such case, because of the precise positioning of grid lattice points required between a nested grid and its parent, CENTLAT and CENTLON will be only approximate, and the precise location of the nested grid will be the closest allowable position to that which they indicate.

NINEST specifies the location within a coarser parent grid where the western edge of a nested grid is to be located. To be specific, we refer to the example in Figure 1. Here, the western edge of grid 2 (the western edge always being defined by the lattice line passing through the u-velocity points having an index i=1) is located along the coarse mesh line passing through the U-velocity points having an index I=3. This configuration would be established by setting the second value (for grid 2) of NINEST to 3. Note that the value of NINEST for a nested fine grid always refers to the I-index in the coarser parent grid of a row of U-velocity points. An important constraint involving NINEST, NNXP, and NSTRATX is that the edge of a fine nested grid cannot be any closer than two cells of the coarser parent grid from the lateral parent grid boundary. The example in Figure 1 is right at this limit on both the eastern and western boundaries of the fine grid: NINEST is set to 3, which places the western boundary of the fine grid just two coarse cells from the coarse-grid western boundary, and the eastern boundary of the fine grid is along the I=5 index row of coarse grid U-velocity points, which is just 2 coarse cells away from the coarse-grid eastern boundary at I=7.

NJNEST corresponds to NINEST, but locates the southern boundary of a fine grid within the coarser parent grid. Here, the J-index of the coarse-grid V-velocity points is what is referred to by the value of NJNEST. In the example of Figure 1, the second value of NJNEST (for grid 2) would be 3. The constraints described for NINEST likewise apply to NJNEST for the y-direction.

NKNEST corresponds to NINEST, but locates the bottom boundary of a fine grid within the coarser parent grid. Here, the K-index of the coarse-grid W-velocity points is what is referred to by the value of NKNEST. Unlike the case for NINEST and NJNEST, NKNEST can be set to 1, if desired. Doing so would cause the nested grid to begin at the ground, like the coarser parent grid. If the fine nested grid is to begin at some height above the ground, NKNEST can be set to a larger number. NKNEST can never be set to 2, however; nor can the combination of NKNEST, NSTRATZ, and NNZP be such that the top of a fine nested grid be located exactly one cell of the coarser parent grid below the top of that parent grid. Another constraint is that NKNEST may not be set to 1 for a nested grid if its direct parent grid does not itself begin at the ground. Very little testing of nesting configurations where a fine nest does not extend to both the ground and the model domain top has been performed. Some tests which have been done have shown that vertical motion impinging on a nesting boundary of differential horizontal resolution cause strong numerical reflections. For this reason, we recommend for most purposes that a fine nested grid begin at the ground and extend to the model top, or that it have identical horizontal grid spacing as its parent, employing nested grid resolution enhancement only in the vertical.

NNSTTOP is a multi-valued, grid-dependent flag to indicate whether each grid extends to the coarse grid top. It must be set to 1 to indicate that a grid does reach the domain top, and set to 0 to indicate that the top boundary of a grid is below the model domain top. In view of the recommendation given under the entry for NKNEST (see in this Section), NNSTTOP should normally be set to 1. Of course, since grid 1 always extends to the model domain top, the first value of NNSTTOP must always be set to 1. The specified values of NNZP, NKNEST, and NSTRATZ will themselves determine where the top of a fine nested grid is in relation to the model domain top. The user must insure that this is in agreement with the setting of NNSTTOP. If not, the model will give a fatal error message and stop.

NNSTBOT corresponds to NNSTTOP, but indicates whether a nested grid has its lower boundary at the lower model boundary (the ground) or above it. NNSTBOT must be set to 1 if the fine nested grid begins at the ground, and 0 if it does not. Similar to the case for NNSTTOP, the model will check for agreement between NNSTBOT and the values of NKNEST, and will stop with a fatal error message if agreement is not met.

GRIDU is a multi-valued, grid-dependent parameter used to control east-west movement of a nested grid. The value assigned to it will be the velocity in meters per second that the grid will be moved. The velocity specified for each grid is always with respect to grid 1 (which cannot move) or equivalently to the fixed ground coordinates. This model feature is used to keep a nested grid surrounding a feature of interest, such as a thunderstorm, which is travelling. For example, if at some time it is found that the storm is centered within grid 3, but the storm is travelling westward at 20 m/s and will soon reach the western boundary of grid 3, GRIDU for grid 3 can be set to -20 m/s which will keep the storm centered within the grid. When a nested grid moves, it does so in discrete horizontal distance increments equal to the grid spacing on the next coarser mesh in which the moving grid is nested. This is done, because the allowable positions the moving grid may have within the coarser parent grid are themselves discrete. When a grid moves, its interior prognostic variables are simply shifted toward the grid's trailing edge the correct number of points (equal to NSTRATX for that grid), the values near the trailing edge are discarded, and the values near the leading edge are newly interpolated from the coarser parent grid. Special care must be taken by the user to make sure that grid 3 will not itself move too close to the boundary of the coarser grid in which it is nested. One remedy if this happens would be to simultaneously move the coarser grid too, as long as it is not grid 1. Set GRIDU to 0 to not activate grid movement. An alternative to moving grids, which has been widely used but which is not as widely applicable, is to Doppler shift the meteorological features by adding a constant wind vector to all wind values in the domain (see descriptions of UMEAN and VMEAN in Section 7.5). However this alternative cannot be used where the ground surface is not horizontally homogeneous. In using the moving grid feature, it is not yet possible to continually fill topography, land percentage, sea surface temperature, and vegetation type from the standard RAMS files on any of the moving grids or any of the grids nested within them. These fields on moving grids must be interpolated from the respective parent grids instead. Thus, in the namelist, set ITOPTFLG, IPCTLFLG, ISSTFLG, and IVEGTFLG all to zero for moving grids. Also, in order to make sure the interpolated values of these fields, plus the additional fields (1) roughness length, (2) vegetation temperature, (3) vegetation moisture, (4) vegetation mixing ratio, (5) bare soil temperature, (6) shaded soil temperature, (7) bare soil moisture, and (8) shaded soil moisture are not overwritten by assignments in ruser.f each time a grid moves, comment out the sections of all relevant IF blocks in ruser.f that fill these values.

GRIDV corresponds to GRIDU, but applies to north-south movement of a nested grid.

Section 7.3 - The $MODEL_FILE_INFO Namelist

The **$MODEL_FILE_INFO** namelist consists primarily of variables which control data input to and data output from the model. The names of files containing these data are specified here, as well as the times during a model simulation when the data are read or written. The information in this namelist also controls some aspects of how input data is to be used.

TIMSTR denotes the time in seconds, minutes, or hours (see description for TIMEUNIT in Section 7.2) already accumulated in a simulation when a history restart of the model is to commence. It must be equal to the cumulative simulation time at which the history file to be used to initialize the current run was written. The time interval obtained by subtracting TIMSTR from TIMMAX (see in Section 7.2) is the actual simulation time that will elapse during the current run. TIMSTR is relevant only if RUNTYPE (see in Section 7.2) is set to 'HISTORY'. If instead a history restart is not being done, TIMSTR is ignored, and the model time counter is initialized at time 0.

HFILIN is a character variable indicating the history file to be read for initializing the current run. The character string placed in this variable must be the actual name of the history file, including the appropriate path if applicable. Further details on the standard syntax of the history file name are given in the description of HFILOUT and HFUNITS in this Section.

IOUTPUT is a flag controlling whether the model will output any history and analysis files, and if so, whether the history files will be written in binary or ASCII. If set to 0, neither type of file will be generated. Setting IOUTPUT to 1 causes the model to generate both types of output files in ASCII. The purpose of a history file is solely to store prognostic model variables and some additional information generated in a model run so that a subsequent model run can be initialized from the file to continue a simulation. A history file written in ASCII is suitable for transferability to other computing platforms, if desired, but is packed to attain an information density near 75% of that in a binary file. There is a small error (close to machine epsilon) introduced in the process of packing data for writing to a history file, and subsequent unpacking of history file data in a history restart of a simulation. This may initially introduce slight negative values in formerly zero microphysical fields, but RAMS removes such zeros before resuming time stepping. A binary history file is written by setting IOUTPUT to 2. This is the preferred setting as it exactly preserves prognostic field values on a history restart. The only disadvantage of a binary history file is that it is difficult to port to different computer platforms, but the need to do this is rare. The purpose of an analysis file is to store model output to be used in producing graphic visualization of the model output. In order for the generation of visual products to be performed on a machine other than the one used for executing the model (this is useful, for example, when the model is run on a supercomputer and the graphical output is produced on a workstation), the analysis file is written in a packed ASCII format (for IOUTPUT set to either 1 or 2) for easy transfer between machines. Full precision of the model variables is not required in generating visualization products, so only 2 or 3 bytes of information are written to the analysis file for each model word in order to minimize file storage space. Each of these 8-bit bytes contains 6 bits of actual information from the model variables, a ratio achieved by translation of model numerical values into a set of 64 different ASCII characters.

HFILOUT is a character variable used to assign an identifying name to the history files output from the model. Only a single character value is assigned to HFILOUT even if more than one history file is to be written in the current run. HFILOUT is used only as a prefix to the one or more file names, making up the first part of each name. The model itself appends the last part of each history file name, according to the time during the simulation when the file is written and the grid number. That is, a separate history file is written for each grid number, as well as for each history write time. In addition, a single history *header* file is written at each history write time. As the model appends both the time and the grid number onto the history file prefix (given by HFILOUT), each file name written during a simulation has a distinct name, but all still share the common prefix. It is useful to include the letter 'h' in the character string specified for this variable to help identify the output file as a history file. HFILOUT is only relevant if IOUTPUT is set to 1 or 2, indicating that history files are to be created. Note that some UNIX machines have a maximum of 14 characters allowable in their filenames. (See also descriptions for HFUNITS and FRQHIS in this Section.)

AFILOUT is a character variable similar to HFILOUT, but applying to the analysis files, rather than the history files, to be output from a model run. It is useful to include the letter 'a' in the character string specified for this variable to help identify the output file as an analysis file.

HFUNITS is a character variable to which should be assigned a value of 's' for seconds, 'm' for minutes, 'h' for hours, or 'd' for dates. For hours, minutes, or seconds, this character is appended onto the names of the history files output from the model, and becomes the last character or suffix in those names. In the history file names, following the prefix (obtained from HFILOUT) and before the suffix (obtained from HFUNITS), a number denoting the actual simulation time of the history file is inserted. The units of this number are either in seconds, minutes, or hours, the choice being made according to the specification of HFUNITS. The user makes this choice depending on how often the history files are to be written, as well as how long the simulation will last. For example, if the simulation lasts for days, and history files are to be output at intervals of 6 hours, HFUNITS should be set to 'h' so that the numbers inserted in the file names will not be too long. On the other hand, if the model is used to perform a microscale simulation lasting only a few minutes, and history files are to be output every 30 seconds, then HFUNITS must be set to 's', or the model will attempt to write two files of the same name (generated during the same minute or hour of the simulation). If HFUNIT is set to 'd', then a file name consisting of the HFILOUT prefix and date string (yyyy-mm-dd-hhmm) will be constructed by the model.

AFUNITS is a character variable similar to HFUNITS, but applying to analysis files, rather than history files, output from the model.

FRQHIS specifies the time interval in seconds at which successive history files are to be output from the model. Every time the model TIME variable [which begins at zero at the start of a new simulation, begins at the value of TIMSTR for a history file start, and counts up to the value of TIMMAX for each run (see descriptions for RUNTYPE, TIMSTR, and TIMMAX)], reaches a multiple of FRQHIS, a model history file is written. In addition, a history file is written out when a model run terminates, even if TIMMAX is not set to an exact integer multiple of FRQHIS, and a history file is written out when a model run begins, even if the TIME variable is not then a multiple of FRQHIS. In this latter case, if the model run is a history start, the initial history file will be written at the same time as the history file which is read to initialize the run. Because this could lead to a file name conflict (the files corresponding to the same simulation time), the upper-case letter 'R' is appended to the initial output history file following the HFUNITS suffix to denote a 'Restart'.

FRQANL is similar to FRQHIS, but specifies the time interval between successive writes of analysis files, rather than history files, are to be output from the model.

INITIAL is a flag specifying how atmospheric fields in the model are to be initialized. If INITIAL is set to 1, the atmospheric variables are initialized horizontally-homogeneously from a single sounding, which the user supplies in namelist $MODEL_SOUND (see in section 7.5). This option is typically used when the model domain is of relatively limited size (a few hundred kilometers across or less), and is required if the model is run in 2-D. Lateral boundaries in this case are open, and are updated based on interior values or are held constant in time. If INITIAL is set to 2, the sounding in namelist $MODEL_SOUND is not used, and instead, the model inputs complete 3-D fields of atmospheric data from *initialization files*, which have been earlier generated by the model from isentropic output from ISAN (see description for RUNTYPE in Section 7.2, and also see Chapters 2 and 5). These fields are used both as initial fields in the model and as time-dependent fields to which the the lateral boundary region of the model is nudged during integration. Further details regarding the use of variable initialization are given in descriptions of several variables in this Section.

VTIME is a multi-valued parameter specifying the time in seconds, minutes, or hours (see description for TIMEUNIT in Section 7.2) relative to the beginning of a simulation (see definition of simulation time in description of TIMMAX in Section 7.2), when each initialization file listed for the VARFIL variable (see description of and VARFIL in this Section and of IFNVAR in Chapter 6) is current. Each model run which has the INITIAL variable set to 2 requires two or more initialization files to be read in. The first initialization file must correspond to a simulation time at or before the time when the run begins, the last must correspond to a simulation time at or after the run ends, and any additional initialization files must be listed chronologically in between. A common practice, when running the model from initialization files, is to specify a set of initialization files spaced at 12-hour intervals, (corresponding to the times of upper air observations). The initialization files are used together in the model in two different ways. The first way is that when a simulation is first initialized (*i.e.*, with RUNTYPE set to 'INITIAL'), the model fields are initialized directly from the first initialization file (thus, the corresponding first value for VTIME must be set to 0). The second way the initialization files are used is to supply a time series of observational data for the atmospheric model to assimilate during execution. These data are obtained by a linear time interpolation between the two initialization files which bracket the current time of the simulation. The model fields are nudged toward the initialization file values according to values specified for TNUDLAT, TNUDTOP, TNUDCENT, NUDLAT, and ZNUDTOP (see in this Section). If INITIAL is set to 1, this parameter is ignored.

VARFIL is a multi-valued character variable specifying the names of all initialization files (including path names if applicable) to be read into the model, for cases when the INITIAL variable (see in this Section) is set to 2. This file name will be that output from a previous model run with RUNTYPE (see in this Section) set to 'MAKEVFILE'. (See especially the description for VTIME in this Section and for IFNVAR in Chapter 6.) If INITIAL is set to 1, this parameter is ignored.

IVWIND currently serves no function in RAMS, but must be set to 0 for initialization of the model from initialization files.

TNUDLAT , **TNUDTOP**, **TNUDCENT**, **ZNUDTOP**, and **NUDLAT** are parameters which control the option of four dimensional data assimilation (4DDA) by Newtonian relaxation (nudging). This 4DDA option requires that the INITIAL variable (see in this section) is set to 2, so that the model is initialized from initialization files (see description for VARFIL in this Section). The initialization files contain time series of gridded horizontal wind, potential temperature, and total water mixing ratio values, usually analyzed from observations or sometimes large scale model forecasts. The atmospheric model solution is relaxed toward the analyzed data during time integration. The strength of the nudging is given by (I-M)/T, where I is an initialization file data value at a particular location, M is the corresponding model value, and T is a user-specified relaxation (e-folding) time scale. RAMS contains a 3-D array of T values for each grid so that the nudging strength can be specified in a customized way by modifying code. However standard distributions of T may be easily defined through the namelist variables TNUDLAT, TNUDTOP, and TNUDCENT, which define relaxation time scales at the lateral boundary, top boundary, and center regions, respectively, of the model domain. The influence of TNUDLAT extends inward from the lateral boundary of the model domain (coarse grid) by a number of grid cells specified by the user in namelist parameter NUDLAT. The influence function (inverse time scale) increases outward parabolically beginning from the parabola vertex located NUDLAT points in from the boundary. The nudging time scale at that vertex and deeper in the interior of the grid is defined by TNUDCENT. Thus, TNUDCENT can be used to specify a lower bound on nudging strength throughout the domain. The influence of TNUDTOP extends downward from the model domain top to a height specified in namelist variable ZNUDTOP. The influence function increases linearly between these two heights, reaching the minimum value defined by TNUDCENT at and below ZNUDTOP. Note that as time scales are inversely proportional to nudging strength, TNUDCENT should have a larger value than TNUDLAT and TNUDTOP for nudging to strengthen toward the lateral and top boundaries. Use of TNUDLAT and NUDLAT is a replacement for the specification of the Davies lateral boundary nudging in 2c and earlier versions of RAMS. Use of TNUDTOP and ZNUDTOP functions analogously to the Rayleigh friction top boundary condition (see descriptions for DISTIM and NUPTS in Section 7.4), but can be used for observational data fields which are not horizontally homogeneous. This provides a means for damping upward-propagating gravity waves which could otherwise reflect off the top model boundary. TNUDCENT is a part of the data assimilation capability in RAMS, and is definable in the namelist in order to conveniently activate domain-wide nudging. Users of the nudging option should experiment with all three nudging time scales to determine the values which work best for a particular application. The only constraint for numerical stability is that none of the time scales can be less than the model timestep, except that *setting TNUDLAT, TNUDTOP, and TNUDCENT to zero turns off the nudging option*. Values of TNUDLAT and TNUDTOP in the range of 100 to 600 seconds correspond to nudging strengths most often used for the Rayleigh friction top boundary condition and in the Davies lateral boundary condition, but substantially larger values in terms of hours may be better for many applications.

FRQPRT specifies the time interval in seconds at which various field values are to be output from the model to the standard output file (see Chapter 5) generated by a model run. The actual fields or portions thereof to be output are specified via several variables in the **$MODEL_PRINT** namelist (see in Section 7.6). The relation between FRQPRT and the model simulation time is the same as for FRQHIS (see in this Section).

FRQIPR specifies the time interval in seconds at which various integral and other diagnostic values computed from model fields are to be output to the standard output file (see Chapter 5) generated by a model run. These values include a domain integral of kinetic energy, a surface integral of pressure, and peak updraft and downdraft speeds in the domain, and are useful for gaining a quick view of the behavior of a simulation or for detecting possible problems with it. The relation between FRQIPR and the model simulation time is the same as for FRQHIS (see in this Section). Integrals are not properly printed when nested grids are present.

FRQIST specifies the time interval in seconds at which integral quantities described under FRQIPR (see in this Section) are stored for output. For example, if FRQIST is set to 600., the standard output file will show values of the integral quantities at intervals of 600 seconds during the simulation. A total of 90 past values are stored for each integral quantity.

ISTPFL is a flag specifying whether and how often a one-line message indicating the value of the model time variable and the elapsed CPU time on the computer is to be output to the standard output file generated from a model run. If set to 0, the message will never be printed out, if set to 1, it will be printed every timestep, if set to 2, it will be output every other timestep, etc. This message can be considered sufficiently useful and informative for the file space it consumes to merit being printed out every timestep. In examining the standard output file while a run is executing, this message informs the user how far the run has progressed, and the CPU time message indicates how much CPU time was spent between messages, which is the CPU time per model timestep if ISTPFL is set to 1. From this value, the user can easily determine how fast the model is running on a particular machine and how much total CPU time will be required for the run.

INITFLD is a flag specifying whether the field values indicated in the **$MODEL_PRINT** namelist (see in Section 7.6) for output to the standard output file are to be output at the initial time of the model run. A value of 0 will cause the values to not be output at the initial time, while a value of 1 will cause them to be output at the initial time.

INPRTFL is a flag controlling whether the namelist variables are to be written to the standard output file. A value of 0 causes them to not be written, and a value of 1 causes them to be written. Writing namelist variables to the output file is useful for recording the parameters of a run with the other standard output, so this flag should normally be set to 1. However, there are cases when it is desirable to keep the output file small (such as when transmitting the file over a low-speed modem), and a value of 0 for this parameter may be preferred.

SFCFILES is single-valued character variable specifying a file prefix (including path information) to be used for outputting the surface characteristic files on a RUNTYPE of 'MAKESFC' or to input the files with the other RUNTYPEs. If RUNTYPE equals 'MAKESFC', then a separate file will be created for each of the NGRIDS grids in the format of SFCFILES.g01, SFCFILES.g02, etc. If RUNTYPE equals any other setting, then these files will attempt to be read to initialize a grid information. If the file for a grid does not exist, its surface characteristics will be interpolated from the next coarser grid. If SFCFILES is set to the special setting of 'NONE', then the surface characteristics will be initialized according to ITOPTFLG, IPCTLFLG, ISSTFLG, and IVEGTFLG. Obviously, don't set SFCFILES to 'NONE' if you have set RUNTYPE to 'MAKESFC'.

ITOPTFLG is a multi-valued, grid-dependent flag specifying how terrain height data is to be obtained or computed for each model grid. If ITOPTFLG is set to 2 for a grid, the model will call subroutine TOPTINIT in the model module **rsurf.f** to assign the terrain height, which by default sets it to zero. To override this default, subroutine TOPTINIT_USER in the file **rsurf.f** is called, and there a user may customize the terrain height field. The **ruser.f** module is specifically intended for user modification when such a need arises (see Section 9.1). Subroutine TOPTINIT_USER is normally constructed with an IF block checking for the grid number (NGRID). The terrain height values must be assigned for the appropriate grid number. It must be pointed out that if nested grids are used, it is required that the terrain heights be compatible between different grids. The compatibility requirement is two-sided: Fine grid values corresponding to a single coarse grid cell must locally average to the coarse grid terrain height value in that cell, and the fine grid values near the lateral boundary of the fine grid must be those which would result from a bi-quadratic interpolation from coarse grid values, according to the standard nesting procedure. The model is designed to make the terrain heights compatible automatically, and will alter the values assigned to a grid in subroutine TOPTINIT or TOPTINIT_USER if they are not already compatible between grids. A special word of caution is in order here. If a fine grid which is to contain fine resolution terrain height data is to be spawned on a history start at a time after the initialization time of a simulation, the terrain height data on the coarser grids must already have been compatible with it in the preceding portion of the simulation. This may cause some model noise problems during that earlier part of the simulation on the portion of the coarser grids where the fine grid will be later inserted due to a lack of smoothness of terrain height data on the coarse grid in that region (see the description for TOPTWVL in this Section). If for a fine grid ITOPTFLG is set to 0, then the terrain heights interpolated directly from the parent grid will be used by default on that fine grid. This will naturally result in relatively smooth terrain height data for the resolution of the fine grid, but is the safest option where the fine grid is to be spawned after the simulation is in progress because it results in no alteration of topographic data on the coarser grids when the spawning occurs. Setting ITOPTFLG to 1 is the most convenient way to acquire terrain height data for a grid, as it reads the data from general purpose terrain height files (see Chapter 5) which can be created for use in most any simulation (for example, a dataset of global coverage). This option is used in conjunction with namelist variables ITOPTFN, SILAVWT, and TOPTWVL (see in this Section). This flag is only used and the initialization performed when RUNTYPE is set to 'MAKESFC' or SFCFILES is set to 'NONE' on other RUNTYPEs.

IPCTLFLG is a multi-valued, grid-dependent flag specifying how the land percentage variable is to be acquired or computed for a grid. Values of 0, 1, or 2 are applicable for this flag, and have the same meaning as for ITOPTFLG (see in Section 7.4). In contrast to the terrain height data controlled by ITOPTFLG, however, the strict compatibility between coarse and fine grid values of land percentage is not required, although it would be rather senseless for coarse and fine grid values to be grossly different. Land percentage is a variable defined for each surface grid cell in the model, and indicates the fraction of the surface in that cell which is land. The remainder of the cell area is assumed to be covered by water, such as lake, ocean, or river surface. For IPCTLFLG set to 2, the model calls subroutine SFCINITF in model module **rsurf.f** which by default sets the land percentage to the value of namelist variable PCTLCON, described in Section 7.4. If for a fine grid IPCTLFLG is set to 0, the land percentage values for that grid will by default be interpolated from its parent grid. For IPCTLFLG set to 1, values for land percentage are interpolated from general purpose land percent data files (see Chapter 5) which can be created for multiple-purpose model runs. This latter option is used in conjunction with namelist variables IPCTLFN and PCTLWVL (see in this Section). To override any of the above options, subroutine SFCINITF_USER in the file **ruser.f** is called, and there a user may customize the land percent field as described above for modifying TOPT in subroutine TOPTININT_USER. This is further discussed in Section 9.1. IPCTLFLG is only used and the initialization performed when RUNTYPE is set to 'MAKESFC' or SFCFILES is set to 'NONE' on other RUNTYPEs.

ISSTFLG is a multi-valued, grid-dependent flag specifying how the sea surface temperature variable is to be acquired or computed for a grid. Values of 0, 1, or 2 are applicable for this flag, and have the same meaning as for ITOPTFLG and IPCTLFLG described above. In contrast to the terrain height data controlled by ITOPTFLG, however, the strict compatibility between coarse and fine grid values of sea surface temperature is not required, although it would be rather senseless for coarse and fine grid values to be grossly different. Sea surface temperature is a variable defined for each surface grid cell in the model, and indicates the the temperature of any surface water in that cell. For ISSTFLG set to 2, the model calls subroutine SSTINIT in model module **rsurf.f** which by default sets the sea surface temperature to the value of namelist variable SEATMP, described in Section 7.4. If for a fine grid ISSTFLG is set to 0, the sea surface temperature values for that grid will by default be interpolated from its parent grid.

For ISSTFLG set to 1, values for sea surface temperature are interpolated from general purpose climatological data files (see Chapter 5) which can be created for multiple-purpose model runs. This latter option is used in conjunction with namelist variables ISSTFN and SSTWVL (see in this Section). To override any of the above options, subroutine SFCINITF_USER in the file **ruser.f** is called, and there a user may customize the sea surface temperature field as described above for modifying TOPT in subroutine TOPTININT_USER. This is further discussed in Section 9.1. This flag is only used and the initialization performed when RUNTYPE is set to 'MAKESFC' or SFCFILES is set to 'NONE' on other RUNTYPEs.

IVEGTFLG is a multi-valued, grid-dependent flag specifying how the vegetation type variable is to be acquired or defined for a grid. Values of 1 or 2 are applicable for this flag. If IVEGTFLG is set to 1, vegetation-type values are obtained from a RAMS-specific database whose name is specified in namelist variable IVEGTFN (see in this section), whereas if IVEGTFLG is set to 2, values are defined in subroutine SFCINITF in the **rsurf.f** module. In this latter case, the value specifed in namelist variable NVGCON (see in Section 7.4) is used by default for the entire domain. (see Section 9.1). (The RAMS-specific database used for IVEGTFLG = 1 is still experimental and was obtained from the U.S.G.S. by special permission, but is expected to be generally available in the near future.) To override any of the above options, subroutine SFCINITF_USER in the file **ruser.f** is called, and there a user may customize the vegetation type field as described above for modifying TOPT in subroutine TOPTININT_USER. This is further discussed in Section 9.1. This flag is only used and the initialization performed when RUNTYPE is set to 'MAKESFC' or SFCFILES is set to 'NONE' on other RUNTYPEs. It should be noted that when IVEGTFLG is set to 1, the implied land percent accompanying the vegetation type will automatically override land percent values.

ITOPTFN is a multi-valued, grid-dependent character variable specifying the family name of the terrain height data file(s), with path names if applicable, to be read in conjunction with namelist variable ITOPTFLG (see in this Section) set to 1. This variable is not the entire file name, but is only set to the prefix denoting the set of files. These files are named according to a convention which identifies the latitude and longitude of their southwest corners. For example, individual files each comprising an area of 1 square geographic degree from a dataset of 1-kilometer-resolution terrain heights which covers the continental United States have names such as U34N112W, indicating a southwest corner at 34 degrees north and 112 degrees west, with the character 'U' identifying the entire dataset. The value specified for ITOPTFN would be 'U' in this case, as the model supplies the remainder of the file name according to the file(s) it needs to read. These terrain height files are written in the same high- information-density ASCII format as the analysis files (see description of the namelist variable IOUTPUT in this Section). Frequently in model applications, fine grids are located within the bounds of a limited-area terrain height dataset, such as the one described above, and can obtain their terrain height information from that dataset, while coarser grids extend outside the high resolution dataset, and utilize a coarser-resolution dataset of global coverage, not needing the high resolution data. This is why ITOPTFN is made grid-dependent.

IPCTLFN is a multi-valued, grid-dependent character variable indicating the name (with path names if applicable) of the file containing land percentage data for initializing each grid. It is similar to namelist variable ITOPTFN (see in this Section).

ISSTFN is a multi-valued, grid-dependent character variable indicating the name (with path names if applicable) of the file containing sea surface temperature data for initializing each grid. It is similar to namelist variable ITOPTFN (see in this Section).

IVEGTFN is a multi-valued, grid-dependent character variable indicating the name (with path names if applicable) of the file containing vegetation type data for initializing each grid. It is similar to namelist variable ITOPTFN (see descriptions for ITOPTFN and IVEGTFLG in this Section).

SILAVWT is a multi-valued, grid-dependent variable specifying the weight given to the *silhouette* average of terrain height in computing terrain height values for each model grid in conjunction with namelist variable ITOPTFLG (see in this Section) set to 1. A weight of (1.0-SILAVWT) is given to the *conventional* average of terrain height. The transfer of terrain height information from the files specified for ITOPTFN is a 3-step process. First, a horizontal interpolation from the data file, which is in latitude-longitude coordinates, to a temporary grid of comparable resolution in cartesian or polar stereographic coordinates (see description of variable IHTRAN in section 7.2) is done. Second, data is averaged from this temporary grid to a lower-resolution temporary grid, to filter out small scale variations which are not desired on the model grid (see description of namelist variable TOPTWVL in this Section). In this second averaging step, both the conventional and silhouette average are performed. The conventional average is a simple summation over all terrain heights on the finer temporary grid, and a division of this sum by the total number of fine grid cells contained in a single coarse-grid cell, to obtain the terrain height to be assigned to that coarse-grid cell. The silhouette average, on the other hand, finds the mean height of the silhouette, as viewed from the east or west, of the set of fine-grid terrain heights contained within a single coarse-grid cell, and the silhouette of the same points as viewed from the north or south, and averages the two mean silhouette heights together. This becomes the computed silhouette height for that coarse-grid cell. While the conventional average preserves total terrain volume above sea level, the silhouette average adds mass by filling in valleys. It is used to maintain the effective mean barrier height that air must rise to when crossing a topographic barrier such as a ridge. The conventional average lowers this barrier height. The value specified for SILAVWT weights the final coarse grid height between these two averages. The third and final step in computing terrain heights for a given model grid is to interpolate the values from the temporary coarse grid to the model grid. (See also the description for TOPTWVL in this Section.)

TOPTWVL is a multi-valued, grid-dependent variable specifying the wavelength, in grid-cell size units, of the smallest horizontal modes of terrain height data which are to be present on a given model grid. It is applicable only for namelist variable ITOPTFLG (see in this Section) set to 1. Referring to the description of namelist variable SILAVWT above, the value of TOPTWVL controls the size of the coarser temporary grid from which terrain heights are interpolated directly to a model grid. The shortest mode which any grid can resolve is that with a wavelength of twice the grid cell size. Hence, if the coarser temporary grid has, for example, twice the cell size of the model grid, no modes shorter than 4 times the model cell size are available from the coarse temporary grid to be interpolated onto the model grid. In this example, TOPTWVL would be set to 4, indicating that shortest mode with respect to the model grid. Because the numerical model does not properly handle the smallest modes resolvable on a grid, it is generally important not to force these modes into the meteorological fields through overly-fine terrain height modes. The 3-step technique described above in the description of SILAVWT is thus useful for achieving both the elimination of undesirable short wavelength modes of terrain height data from the model grid, and adequate silhouette averaging where desired.

PCTLWVL is a multi-valued, grid-dependent variable corresponding to TOPTWVL (see in this Section), but applying to land percentage data (see description for IPCTLFLG in this Section) rather than to terrain height data. Two differences between land percentage and terrain height data are that silhouette averaging is not applicable for land percentages, and small-mode signals in land percentage data are not normally a problem for a model simulation.

SSTWVL is a multi-valued, grid-dependent variable corresponding to TOPTWVL (see in this Section), but applying to sea surface temperature data (see description for ISSTFLG in this Section) rather than to terrain height data. Two differences between sea surface temperature and terrain height data are that silhouette averaging is not applicable for sea surface temperatures, and small-mode signals in sea surface temperature data are not normally a problem for a model simulation.

MKCOLTAB is a single-valued flag indicating whether or not a *collection table* is to be created in the present run. The collection table is required in the new microphysics package, and contains terms derived from the stochastic collection equation. These terms are computationally intensive to evaluate, and are thus pre-computed and stored in the collection table where they are efficiently accessed during model runtime. The values in the table depend on two sets of parameters. One is the set of gamma distribution shape parameters, defined in namelist variable GNU (see in section 7.5). The other is a list of parameters defined in a data statement in subroutine MICINIT in the file rmcth.f, which describe the relation between mass and diameter, and between fall velocity and diameter for the hydrometeor categories. These latter parameters are not normally altered, unless a user has reason to change these relationships. The shape parameters defined in the namelist, however, are intended to be adjusted and experimented with. Any change in any parameter of either set requires a different collection table to be created and used. Setting MKCOLTAB to 1 causes the table to be created at the beginning of the present run, and to be stored in a file of the name given in namelist variable COLTABFN (see below). The relevant parameters from both the namelist (the GNU variable) and the data statement in the code) are written in the first lines of the collection table. If MKCOLTAB is set to 0, the model will not create the table, but will instead attempt to read the file specified in the variable COLTABFN. This option avoids the need to generate the collection table each time a run is made. When the collection table is read in, the list of parameters in its first few lines will override the parameter settings in the namelist and the data statement.

COLTABFN is a single-valued character variable giving the name (and path, if any) of the *collection table*, described above.

EVPTABFN is a single-valued character variable giving the name (and path, if any) if the *evaporation table*. This table is required by the new microphysics package, and is currently generated by a stand-alone program. Its values are a function only of the gamma distribution shape parameter for pristine ice, which is the third value of GNU specified in the namelist (see below). Collection tables for integer values of GNU from 1 to 5 are currently supplied, under the names ct1.0, ct2.0, ct3.0, ct4.0, and ct5.0. These will eventually be generated within RAMS.

Section 7.4 - The $MODEL_OPTIONS Namelist

The **$MODEL_OPTIONS** namelist is where the majority of choices for specifying model parameterization options are made. The variables in this namelist are those which do not fit under the special categories of the **$MODEL_GRIDS** and **$MODEL_FILE_INFO** namelists, *i.e.*, those which neither control grid configuration, space and time domains, nor the reading or writing of data files.

NADDSC is a multi-valued, grid dependent parameter specifying a number of prognostic scalar fields to be added to the model simulation. Such scalar fields might be desired by the user for representing quantities such as chemical or aerosol pollutants. The added scalar fields are automatically advected, diffused, and marched forward in time, simply by setting NADDSC to a positive value. The new variable names that are added to to model are SCLP(1), SCLP(2), etc. for the "past" time level, SCLC(1), SCLC(2), etc. for the "current" time level, and SCLT(1), SCLT(2), etc. for the tendency arrays. These, therefore, may be refenced in the "A" array as A(ISCLP(1)), A(ISCLP(2)), etc. It is up to the user to add routines in the code to compute any initialization, source or sink terms required for the added scalars. This procedure is described in Section 9.3. The number of scalars that can be added in this way is unlimited (provided computer memory is large enough), but the value of the parameter MAXSCLR described in Section 7.1 must be greater than or equal to NADDSC.

NTOPSMTH controls a simple 1-2-1 type filter used in smoothing terrain height data on the coarse grid. The value of NTOPSMTH specifies the number of times this filter is successively applied to the data (for no smoothing, set it to 0). This filter should not be used if smoothing has already been performed through the use of TOPTWVL (see in this Section). The purpose of NTOPSMTH is to perform any necessary smoothing to a terrain height dataset which is input to the model using ITOPTFLG (see in this Section) set to 2.

IZFLAT is used to alter terrain heights near the lateral boundaries of the coarse grid such that they are held constant in the direction normal to the local lateral boundary. The value of IZFLAT specifies the width of the zone in number of grid cells where this is done (a value of zero causes no flattening to be done). For example, if IZFLAT is set to 3, the terrain height value at any location two grid points in from a lateral boundary, such as at I=3, is assigned to the two grid points outside (I=1 and 2). We do not often activate this feature as it does not seem to be generally required.

ITMDIFF is a flag specifying the type of time differencing to be used in the run. A value of 1 specifies forward time differencing for all variables, requiring only one time level. Setting ITMDIFF to 2 causes leapfrog time differencing to be used for all variables, in which case two time levels are used requiring more computer memory. Setting ITMDIFF to 3 activates a hybrid of the above two choices wherein velocity components and pressure are updated using leapfrog differencing, and all other prognostic variables are advanced using forward differencing. Directly tied to the form of time differencing is the advection algorithm used, and the so-called "forward" and "leapfrog" advection schemes used in RAMS have somewhat different performance characteristics in terms of damping and dispersion. If NONHYD (see this Section) is set to 0 (a hydrostatic run), only forward time differencing (ITMDIFF=1) is currently available. The hybrid time differencing is starting to now become the scheme that is most often used for the nonhydrostatic model.

NONHYD is a single-valued flag specifying whether the model is to be run nonhydrostatically or not. A value of 1 for this flag makes the model run in nonhydrostatic mode, and a value of 0 makes it hydrostatic. One advantage to running the hydrostatic option is that for coarse horizontal grid spacing, that is, larger than 40 or 50 kilometers, a timestep longer than the 90 second limit for the nonhydrostatic model may be used, causing the model to run faster. Both the hydrostatic and nonhydrostatic options give virtually the same results on coarse grids. Where high resolution is required, for example in simulations involving deep convection or flow over a significant topographic feature, the hydrostatic assumption is not valid and the nonhydrostatic option must be used. Since all grids in a given simulation must follow the same option, if a fine nested grid requires the nonhydrostatic option, the coarser grids must use it as well.

SSPCT specifies the ratio between the effective speed of sound which the model will use in simulating the propagation of sound waves, and the physical speed of sound computed from the local temperature. If set to 1.0, the model speed is the same as the physical speed, and if set to 0.5, for example, the model speed will be half the physical value. The model sound speed is slowed by increasing the bulk modulus of elasticity, or equivalently, by reducing the amount the pressure in a fixed volume increases when a given amount of mass is added adiabatically to it. Sound speed reduction is a useful means of increasing the efficiency of the model integration, given the explicit numerical treatment of sound waves in the model formulation. Experience shows that as long as the model sound speed is at least a factor of 2 or 3 faster than characteristic velocities of other disturbances (whether advective or gravity wave speeds are more important here probably depends on the particular model configuration), the model solution is very little effected by the sound speed reduction. In practice, we commonly reduce the model sound speed to 1/3 or 1/2 its physical value. Numerical stability requires the model sound speed to be slow enough for the horizontal grid spacing and the length of the small timestep, meaning that SSPCT, NRATIO, DTLONG, and the smaller of DELTAX or DELTAY (see descriptions for the latter four in Section 7.2) must together satisfy a constraint. The user may set SSPCT to 0 in which case the model will compute an appropriate value of SSPCT based on these other parameters. SSPCT is relevant only for nonhydrostatic model runs.

IMPL is a flag specifying whether the computations which specifically involve propagation of sound waves in the model will be performed semi-implicitly in time in the vertical direction. A value of 1 activates the implicit calculation, while a value of 0 causes the calculations to be performed explicitly in the vertical, as they are always done in the horizontal. The implicit option is made available in the vertical since in most simulations, the vertical grid spacing is much lower than the horizontal spacing, and explicit sound wave computation would require an unreasonably small timestep. Thus, IMPL is nearly always set to 1. In simulations where the vertical and horizontal grid spacings are comparable, the user may wish to set IMPL to 0 for slightly greater efficiency, but there is some indication that the semi-implicit scheme actually gives a better solution. This parameter is relevant only for nonhydrostatic runs.

ICNTEQ is a parameter applying only to the hydrostatic option in the model (see description for NONHYD in this Section), and specifies whether vertical velocity is to be evaluated from horizontal divergence assuming incompressibility or anelasticity in the vertical integration of the continuity equation . A value of 1 activates the incompressible assumption, while a value of 2 activates the anelastic assumption. The latter is normally preferred, as it is more physically realistic. For the nonhydrostatic option, a fully elastic continuity equation is always used.

WTKD specifies the weighting given to the Klemp-Durran (K-D) upper boundary condition (see description for IBCTOP in this Section). The allowable range for this parameter is 0 to 1, with 0 giving no weighting and 1 giving full weighting to the K-D condition. The K-D condition is a computation for pressure at the upper boundary of the model. For nonhydrostatic runs, zero weighting of the K-D pressure leaves the pressure there entirely specified by extrapolation from prognosed values below, while for hydrostatic runs, the upper boundary pressure remains constant in time. WTKD is only used as a parameter if IBCTOP is set to 1.

ICORFLG is a flag specifying whether the Coriolis force is to be activated, and, if the simulation is 2-D, whether the v-component of velocity is to be activated. A value of 0 causes the Coriolis force to not be used. In a 2-D simulation where only the u and w velocity components are in the grid plane, since the v-component would be uncoupled from u and w without Coriolis force, a value of 0 for ICORFLG deactivates prognosis or use of the v-component. If the simulation is horizontally homogeneous (namelist variable INITIAL, see in Section 7.3, set to 1), no horizontal pressure gradients are initially available to balance the Coriolis force, and it is useful to compute the Coriolis force only from the difference between the current model winds and a reference state wind (normally the initial model wind interpolated from the sounding). Then, the true horizontal distribution of pressure would be regarded as the sum of the prognosed model pressure and a gradient in geostrophic balance with the reference state wind.

IBCTOP is a flag specifying the type of upper boundary condition to be used in a simulation. A value of 0 applies only to the nonhydrostatic configuration of the model, and activates a rigid lid at the model top by constraining vertical velocity to be zero there. This condition is simple, but has the disadvantage of causing strong reflection of upwardly propagating gravity waves, turning them downward back into the model domain. For many simulations, this is not a significant problem, but for some, such as a commonly performed simulation of terrain-induced gravity waves over a low barrier, the rigid lid totally corrupts the results. In cases such as these, if a rigid lid is to be used, a Rayleigh friction absorbing layer (see descriptions of NFPT and DISTIM in this Section) may be activated in the upper layers of the model to absorb gravity waves approaching the lid, sufficiently damping them before and after reflection that they are effectively eliminated. A value of 1 for IBCTOP activates the Klemp-Durran (K-D) top boundary condition, which is specifically suited to allowing internal gravity waves to propagate out of the model top. The K-D condition is a computation for perturbation pressure at the model top, based on the vertical velocity there which is prognosed by the model, assuming that perturbation pressure and vertical velocity are related according to the motions of pure steady-state gravity waves. This condition works very well for the mountain wave simulations described above, but is not properly formulated for other types of disturbances in which the physical relation between perturbation pressure and vertical velocity is different. A well-known example where the K-D condition tends to fail is in a situation of strong convection. The K-D condition may be used for either the hydrostatic or nonhydrostatic model configuration. However, it is not appropriate to use it for variable initialization (the INITIAL variable, see in Section 7.3, set to 2), as such initialization normally carries with it large scale gradients of pressure and vertical motion which are totally incompatible with the relationship assumed by the K-D condition. A value of 2 for IBCTOP only applies to hydrostatic simulations, and is a top boundary condition which specifies that the horizontal divergence above the model top is zero. This option causes the surface pressure to be prognosed, based on the vertically-integrated horizontal mass divergence in the column above, and the pressure is computed by vertical integration starting at the surface. A value of 3 for IBCTOP applies only to the hydrostatic configuration of the model, and activates a computation of pressure at the top boundary based on the vertical velocity and an assumed advection of a material surface. This condition is neither recommended nor currently supported.

IBND is a flag controlling the lateral boundary conditions (in the x-direction) applied on the coarse grid only to the normal velocity component (except for IBND = 4). Values of 1, 2, and 3 for this flag each activate a condition designed to allow disturbances to propagate out of the domain. A value of 1 activates the Klemp-Wilhelmson condition in which the normal velocity component specified at the lateral boundary is effectively advected from the interior assuming a propagation speed (intended to be similar to a dominant gravity wave phase speed) specified in the namelist in the variable CPHAS (see in this Section). This condition is the one most often used in RAMS, and is recommended. A value of 2 for IBND activates the Klemp-Lilly condition, which is a variant of the Orlanski condition (IBND = 3) in which the gravity wave propagation speeds computed for each grid cell in the Orlanski condition are averaged vertically, with the single average value being applied over the entire vertical column. A value of 3 for IBND activates the Orlanski condition in which a gravity wave propagation speed is computed for each grid cell at the lateral boundaries based on the apparent propagation speed of disturbances approaching the boundary from the interior. Setting IBND to 4 sets up a cyclic boundary condition between the two x-direction boundaries wherein all prognostic variables in the model, not just the normal velocity component (u), are assigned at the boundaries from values taken from the model interior just inside the opposite boundary. This makes the model periodic in the x-direction, effectively eliminating lateral boundaries from the solution. The number of grid points in the x-direction comprising one complete cycle in the periodic field is 3 less than NNXP (see in Section 7.2) for the coarse grid, if the second-order advection scheme is used (see descriptions for IADVL and IADVF in this Section). For the 4th- and 6th-order advection schemes, the periodic cycle length is 5 and 7 points, respectively, less than NNXP for the coarse grid.

JBND is similar to IBND, but applies to the y-direction, rather then x-direction boundaries. In a 2-D simulation, JBND is irrelevant.

CPHAS is used in conjunction with IBND and/or JBND (see in this Section) set to 1, and specifies the characteristic propagation speed of internal gravity waves used in the Klemp-Wilhelmson lateral boundary condition. A value of 20 m/s is a common setting for this parameter. Use of larger values, even very much larger, still allows propagation of gravity waves out of the domain reasonably well, whereas much smaller values tend to lead to stronger reflection of the waves back into the domain interior. A very large value of CPHAS is equivalent to the zero-gradient boundary condition.

LSFLG complements IBND and JBND (see in this Section) as a lateral boundary condition flag, applying to all variables other than the normal velocity component. It deals with all these remaining variables in the same way. A value of 0 for LSFLG sets the lateral boundary value of each variable to the value in the field immediately adjacent to the boundary in the interior. This is a zero-gradient condition. This option is the most commonly used for horizontally-homogeneous simulations (the INITIAL variable, see in Section 7.3, set to 1), and is recommended as a first-try option. Occasionally at inflow boundaries, this zero-gradient option has been found to cause a runaway trend to a field variable, and must then be replaced by another option. A value of 1 for LSFLG is very similar to the 0 value, using a zero-gradient condition at inflow boundaries, but setting values at outflow boundaries such that the second spatial derivative in the direction normal to the boundary is zero. This is simply an extrapolation from two interior points. This condition is superior for some cases to the simpler zero-gradient option at the outflow boundary, but usually makes no significant difference. A value of 2 for LSFLG is identical with the value of 1 for the outflow boundary condition, but holds variables constant in time at inflow boundaries. This is a possible remedy to the runaway problem sometimes encountered at inflow boundaries, as described above, but is not without its own shortcomings. In cases where flow reverses in time at a lateral boundary, alternating between inflow and outflow, option 2 for LSFLG will cause the boundary condition to alternate between constant in time and time-variant. This may generate an undesirable model response in some cases. Another problem with holding the variables constant in time at an inflow boundary is that those constant values are continuously advected into the model interior, and may become incompatible with the evolving situation in the domain. For example, if a boundary layer is growing in the model in response to solar heating, the constant-in-time inflow condition will cause the initial cooler values of temperature to flow into the model from the boundary, artificially cooling the boundary layer. A good solution for this situation is to directly specify an appropriate temporal evolution of the boundary value, still using the LSFLG = 2 option. This is not a presently a standard feature in the code, but can easily be implemented by the user in subroutine LATSET, located in the module **rbnd.f**. A value of 3 for LSFLG leaves values at all boundaries, inflow and outflow, unchanged in time. This option can be used only for variable initialization simulations (the INITIAL variable, see in Section 7.3, set to 2), and has sometimes, but not always, been found to give better results than LSFLG = 0 for INITIAL = 2.

NFPT specifies the number of grid levels on the coarse grid, starting from the top of the model domain, in which the Rayleigh friction absorbing layer will be activated. (See also the descriptions for IBCTOP and DISTIM in this Section.) A value of 0 leaves Rayleigh friction inactive. Rayleigh friction must be applied only in stable air, and thus requires the model domain to extend sufficiently high, most commonly in the stratosphere. The bottom layer of the Rayleigh friction zone should be above the top of whatever meteorological system is important to the simulation, so as not to directly interfere with its behavior. This normally requires adding more model levels in the vertical than would otherwise be used in a simulation not employing Rayleigh friction. Rayleigh friction is a relaxation of all 3 velocity components (only u and v in the hydrostatic configuration) and potential temperature toward the undisturbed, horizontally homogeneous reference state values. Its purpose is to damp gravity wave and other disturbances which approach the top model boundary, so that they will not be reflected back downward, and is normally used in conjunction with a rigid lid to the model (IBCTOP, see in this Section, set to 0). Because prognostic fields which are not initialized horizontally homogeneously cannot sensibly be forced toward a horizontally-homogeneous state, Rayleigh friction must not be used with namelist variable INITIAL (see in Section 7.3) set to 2. However, top boundary nudging (which damps gravity waves) for horizontally-inhomogeneous initialization may be activated elsewhere (see the descriptions for TNUDTOP and ZNUDTOP Section 7.3.)

DISTIM is used in conjunction with namelist variable NFPT (see in this Section) in activating Rayleigh friction in the upper region of the model. DISTIM specifies a dissipation time scale in seconds, representing the e-folding decay time of a disturbance being damped by the Rayleigh friction layer in the absence of any other source or sink terms. This decay rate strictly applies at the top boundary of the model only, and the strength of the damping decreases linearly with height, reaching zero at the lowest of the NFPT levels in which the damping is activated. DISTIM should be related to the rate at which the model is generating upward propagating gravity waves, and normally ranges from 60 to 200 seconds. DISTIM must be at least twice the value of DTLONG (see in Section 7.2) for the nonhydrostatic option or greater than DTLONG for the hydrostatic option for numerical stability. The model solution proves to be rather insensitive to DISTIM, provided it is in or near this range.

TIMSCL is used in conjunction with namelist variables KSPIN and KMSPIN (see in this Section) in gradually spinning up model winds from zero to some observed value in the initial minutes of model integration. TIMSCL specifies the length of time in seconds over which this spin-up process, which is linear in time, takes place. This gradual increase of winds has sometimes been found necessary in cases where strong horizontal winds encounter significant topographic barriers, because otherwise the initial shock to the model (vertical velocity is normally zero initially) is too large. A more important need arises from the fact that initial horizontal winds in a domain with topography usually are not adjusted so as to prevent net mass flux into or out of the domain (see the description for KMSPIN in this Section). A value of 1000 seconds or so is reasonable for TIMSCL in a typical mesoscale simulation. A value of 0 will cause the spin up to not be performed. The spin-up option should normally not be used unless a test without it demonstrates that is is required.

KSPIN is used in conjunction with namelist variable TIMSCL (see in this Section) and specifies the highest level in the model below which the spin-up of the initial winds will take place. Above the level KSPIN, the winds will start at full strength.

KSPIN should normally be a sigma level whose geometric height over the lowest terrain in the domain is well above the highest terrain in the model.

KMSPIN is used in conjunction with namelist variables TIMSCL and KSPIN (see in this Section) and specifies a highest level in the model below which a special adjustment of wind speed, designed to conserve mass in the domain, is made. This special adjustment intended for use when the terrain heights at the inflow boundaries are substantially different and strong winds are flowing at low levels over the surface. If those winds are not sufficiently varied in strength horizontally, the mass flux into and out of the domain will be unequal, and domain-wide mass trending will result. Use of KMSPIN causes the winds over the higher terrain to be made stronger than those over the lower terrain during the spin-up process such that zero net mass flux to or from the model domain is established. The value of KMSPIN should reflect the approximate height above the terrain up to which significant variations in mean wind speed induced by terrain are felt. A vertical distance comparable to the height of the terrain barrier may be appropriate.

IPRSPLT is a flag specifying whether a time-split scheme is to be used for simulating the fall of precipitation. It would be required if the precipitation velocity is so large that the downward displacement is greater than one model level in a timestep, thus violating the CFL stability criterion. A value of 0 for IPRSPLT does not activate the time split scheme, instead artificially reducing the fall velocity to the stability limit. A value of 1 employs a time-split procedure in which the precipitation process is simulated on two or more shorter timesteps at the full diagnosed velocity. This latter option is not currently implemented in the nonhydrostatic version.

IADVL is a flag specifying the order of accuracy of advective scheme to be used for any prognostic fields advanced using leap-frog time differencing (see description of ITMDIFF in this Section). Allowable values for this flag are 2 (for second-order advection) and 4 (for fourth-order advection). The most common choice for this parameter has been 2, but the model solution is not generally highly sensitive to the order chosen. The higher order schemes are theoretically more accurate, but require a bit more computation time.

IADVF is similar to IADVL, but applies to any prognostic variables advanced in time using forward (rather than leapfrog) time differencing. Allowable values for this flag are 2 (for second-order advection) and 6 (for sixth-order advection). The most common choice for this parameter has been 2, but the model solution is not generally highly sensitive to the order chosen. The higher order schemes are theoretically more accurate, but require a bit more computation time.

IPGRAD is a flag specifying how horizontal pressure gradients are to be evaluated in the model. The flag is relevant only where variable terrain height is used at the surface. A value of 1 for IPGRAD evaluates the horizontal pressure gradient from an appropriate weighting between the vertical pressure gradient and the gradient along a sloping sigma-z surface. This is the option almost always used, and is recommended. A value of 2 for IPGRAD evaluates the horizontal pressure gradient between a local pressure and an adjacent pressure at the same geometric level which has been vertically interpolated from levels where the pressure is defined. This latter option is only relevant for the hydrostatic version of the model.

FILT4 specifies whether a fourth-order filter is to be used to horizontally filter prognostic fields in the model, and if so, at what strength. A value of 0 results in the filter not being used, and a value greater than zero specifies a time in seconds in which the filter will attenuate by a factor of e a disturbance of wavelength $2\ \Delta x$ on the grid. In other words, a small value of FILT4 implies strong filtering. The value of FILT4 must be larger than twice the value of DTLONG (see in Section 7.2) for the nonhydrostatic option and greater than DTLONG for the hydrostatic option for numerical stability. The fourth-order filter has undesirable properties where terrain has a significant slope, but executes considerably faster than the normal horizontal diffusion. When normal horizontal diffusion is activated in the model (see description for IDIFFK in this Section), the fourth-order filter should not normally be used.

FXLONG specifies whether an alternative to the fourth-order filter (see description for FILT4 in this Section), known as Long's filter, is to be activated in a simulation, and if so, at what strength. It is more expensive than the fourth-order filter and does not perform better, so its use is not recommended. FXLONG controls filtering in the x-direction.

FYLONG is similar to FXLONG, but applies to the y-direction instead of the x-direction.

ISWRTYP is a flag specifying one of two independent shortwave radiation parameterizations to be used in a simulation. A value of 0 turns off shortwave radiation while a value of 2 activates a scheme described by Mahrer and Pielke. It requires relatively little computational expense, but does not include the radiative effects of liquid water or ice. A value of 1 for ISWRTYP activates a scheme developed by Chen and Cotton. This scheme is more expensive computationally than Mahrer and Pielke to run, but accounts for the radiative effects of liquid water and ice. (See also descriptions for ILWRTYP and RADFRQ in this Section.)

ILWRTYP is similar to ISWRTYP (see in this Section), but applies to longwave, rather than shortwave, radiation.

RADFRQ specifies how often during a model run the radiative parameterization is to be exercised to compute updated values for the radiative contribution to tendencies of both atmospheric and surface soil temperatures. These tendencies are applied every timestep of the model integration, but the tendencies do not normally need to be updated that often. In fact, because computing the radiative contribution to temperature tendencies is a relatively computationally expensive process, it is strongly suggested that this computation be done only at intervals of 600 to 1200 seconds (by specifying such values for RADFRQ). In most situations, this is often enough to account for changes in radiative transfer.

LONRAD is a flag specifying whether the longitudinal variation of solar hour angle is to be accounted for in the computation of shortwave solar radiation, or whether these angles are to be assumed constant over the model domain. A value of 0 for LONRAD assumes the hour angle is longitudinally constant, and a value of 1 accounts for its longitudinal variation. The latter choice is the one normally made in a simulation of a real event, while the former choice is used when a certain level of idealism is required in a simulation, such as horizontal homogeneity of external forcing.

NNQPARM is a multi-valued, grid-dependent flag used to control whether the convective parameterization is to be activated or not. A value of 1 for this flag activates the parameterization, and a value of 0 leaves it inactive. Convective parameterization is used to vertically redistribute heat and moisture (as if by convection) in a grid column when the model generates a region which is superadiabatic or convectively unstable and when the horizontal grid resolution is too coarse for the model to develop its own convective circulation. Ideally, resolving a convective circulation would require at least a few grid cells to horizontally span an updraft, which for deep convection would normally require the horizontal cell size to be less than 1 or 2 kilometers. Coarser resolution than this would make realization of sufficiently strong vertical motion difficult or impossible to adequately bring about the required vertical exchange of heat and moisture so as to convert the convective available potential energy into other forms. Thus, it is on coarser grids where a parameterized convective adjustment becomes necessary. Unfortunately, the convective parameterization schemes currently available assume the grid size in the horizontal to be around 20 kilometers or greater. This means that convective parameterization may be activated on any grid of this resolution, but that at resolutions between about 2 and 20 kilometers, no adequate convective adjustment scheme exists. A new convective parameterization scheme intended to fill this resolution gap is currently undergoing extensive testing in RAMS.

CONFRQ specifies the frequency in seconds during a simulation at which contributions to atmospheric tendencies in temperature and moisture by convective parameterization are to be computed. These tendencies are applied to the temperature and moisture fields every timestep of the model integration, but it is not required that the convective parameterization contribution to these tendencies be updated every timestep. In fact there are two reasons why those contributions should not be updated so frequently. One is that a computational expense is involved in exercising the convective parameterization, and frequent exercise thus slows down the model. The change in the tendencies would not be large from one timestep to the next anyway. A second reason has an important physical basis. For example, consider a case of a squall line moving through a grid volume. As the squall line enters the grid volume, it will begin to stabilize the air that it has moved through. However, the air in front of the squall line is still convectively unstable and will contribute to the maintenance of the squall line. The mean atmosphere *i.e.*, that represented on a coarse grid volume, soon may be stabilized by the convection. If the convective tendencies were to be re-evaluated in the model each 5 minutes or so, the result for the above example would be that the original convection might be shut down in its early stages. A more accurate representation of the squall line, however, would be to let it run through the entire grid volume. This is achieved by setting CONFRQ to a time scale comparable to the length of that life cycle, such as 1200 to 1800 seconds. CONFRQ is relevant only if the parameter NNQPARM (see in this Section) is set to 1 for a particular grid.

WCLDBS is used in conjunction with NNQPARM and CONFRQ (see in this Section), and specifies the threshold value of vertical velocity in m/s in the model at the diagnosed level of cloud base required to initiate convection. The more positive this parameter, the more difficult is the initiation of convection. Values for this parameter should normally range from about -0.1 to 0.2.

ISFCL is a flag specifying which of 2 possible options for obtaining surface fluxes of heat and moisture to or from the atmosphere will be used in a simulation. If ISFCL is set to 0, the user specifies values for namelist variables DTHCON and DRTCON (see in this Section) representing the result of subtracting a fictitious ground temperature and moisture mixing ratio, respectively, from the surface atmospheric values of the two variables. These differences are then converted to vertical fluxes of heat and moisture to or from the lowest atmospheric level. This is the simplest of the 2 choices, and ignores the soil model parameterizations available in the model. A value of 1 for ISFCL activates a soil model developed by Tremback and Kessler and a vegetation model based on Avissar. This soil model prognoses soil and vegetation temperature and moisture based on vertical diffusion and exchange with the atmosphere of both quantities. (See also descriptions for namelist variables NNZG, PCTLCON, NSLCON, SOILDZ, SLZ, and STGOFF in this Section.)

NVGCON is a flag specifying the type of vegetation covering the ground surface. By default, the value of this flag is used to fill all surface grid cells. If it is desired that vegetation type be spatially varying rather than homogeneous, this default may be overridden in the file **ruser.f** (see section 9.1). Values ranging from 1 to 18 are allowed for NVGCON, and have the following meanings:

```
 1 -- Crop/mixed farming                10 -- Irrigated crop
 2 -- Short grass                       11 -- Semi-desert
 3 -- Evergreen needleleaf tree         12 -- Ice cap/glacier
 4 -- Deciduous needleleaf tree         13 -- Bog or marsh
 5 -- Deciduous broadleaf tree          14 -- Inland water
 6 -- Evergreen broadleaf tree          15 -- Ocean
 7 -- Tall grass                        16 -- Evergreen shrub
 8 -- Desert                            17 -- Deciduous shrub
 9 -- Tundra                            18 -- Mixed woodland
```

These classifications are based on the Land Cover/Vegetation Type from the Biosphere-Atmosphere Transfer Scheme (BATS). Note that setting NVGCON to 8 is equivalent to running without vegetation parameterization.

TSEASN is a parameter specifying an average seasonal temperature which is used to determine the *leaf area index* and *fractional coverage* of vegetation, which are seasonally-varying parameters. Use of this parameter is based on the BATS scheme, which assumes that the leaf area index and fractional coverage vary quadratically with TSEASN, with maximum values occurring at TSEASN = 298 K. The actual quadratic functions are defined in subroutine SFCDATA in the file **rsurf.f**.

TVGOFF is a parameter used in initializing vegetation temperature at the beginning of a simulation. It represents the number of degrees (K) above the initial local air temperature that the initial local vegetation temperature is set. By default, the value set by the user for TVGOFF is applied to all surface locations in the simulation. If it is desired that the actual offset be spatially-variable, the default may be overridden in the file **ruser.f** (see Section 9.1).

VWTCON is not yet utilized in the model, and should not be changed (leave it set to 0).

UBMIN specifies a minimum value of horizontal velocity in the lowest level of the atmospheric model grid to be used in computing the local friction velocity u_*, and is a means of imposing a minimum value on friction velocity It is used in the computation of surface fluxes of momentum, heat, and moisture, which is done by surface layer similarity theory from given atmospheric and soil conditions. The ability to specify this minimum value greater than zero has been made an option in the model based on the argument that if even the resolved wind in the lowest model level becomes very close to zero, which would otherwise result in a computation of zero for the friction velocity, the normal subgrid eddies would still have a characteristic mean wind speed in variable directions causing the effective friction velocity to be positive.

PCTLCON specifies the fraction, ranging from 0 to 1, of a surface grid cell which is covered by land surface, implying that the remainder is covered by water. This distinction determines the values of surface fluxes of momentum, heat, and moisture that are exchanged between the atmosphere and surface. Separate calculations of these fluxes are made for each grid cell for the soil and water surfaces, and the resultant net flux is weighted between these two according to the relative area covered by each type of surface. By default, the value assigned to PCTLCON is assigned to each surface grid cell in the model. If, as is commonly the case, the user wishes to define a horizontally variable distribution of land percent values, the subroutine

SFCINITF_USER in the file **ruser.f** needs to be modified (see Section 9.1), or land percentage values can be read in from a standard file (see description for IPCTLFLG in Section 7.3).

NSLCON specifies the type of soil used in the soil model. The soil type controls important soil properties which ultimately affect the influence by the soil on the atmosphere. These properties are the thermal diffusivity, the heat capacity, the capacity of the soil to hold moisture, and the diffusivity of moisture. The thermal properties are themselves highly influenced by the moisture content of the soil, but the particular relation between them is dependent on the soil type. A total of 12 different soil types are parameterized in the model. The following is a list of each soil type and the value of NSLCON which activates it: sand - 1, loamy sand - 2, sandy loam - 3, silt loam - 4, loam - 5, sandy clay loam - 6, silty clay loam - 7, slay loam - 8, sandy clay - 9, silty clay - 10, clay - 11, peat - 12. As in the case for PCTLCON (see in this Section), the value assigned to NSLCON in the namelist is in turn assigned to the soil type for each surface grid cell in the model. If it is desired to make the soil type horizontally inhomogeneous, the user must modify subroutine SFCINIT_USER in the module **ruser.f** (see Section 9.1).

ZROUGH specifies the roughness length, a parameter important in representing the effects of ground roughness on the surface fluxes of momentum, heat, and moisture. The parameterization of these fluxes is designed according to surface similarity theory, in which the roughness length is a crucial parameter. A roughness length of 0.01 meter or less represents a smooth surface, while a roughness length of 1.0 meter represents a very rough surface such as a forest of tall trees. As in the case of PCTLCON (see in Section 7.3), the value assigned to ZROUGH in the namelist is in turn assigned to each grid point in the model. Should the user wish to make roughness length a location-dependent parameter, subroutine SFCINIT_USER in module **rsurf.f** must be modified (see Section 9.1). ZROUGH only refers to the soil areas of the model surface; the vegetation roughness lengths vary with type and are set in the SFCDATA subroutine.

ALBEDO represents the albedo or reflectivity of the ground surface to solar radiation. This parameter only applies when the soil model is not activated, *i.e.*, for ISFCL (see in this Section) set to 0. In this case, the albedo is not involved in the computation of surface sensible and latent heat fluxes; it is only used to compute solar radiation reflected upward from the ground, which is needed in the atmospheric radiation budget.

SEATMP specifies the temperature in Kelvins of the surface water temperature. As for other namelist parameters such as PCTLCON (see in Section 7.3), the value specified for SEATMP is by default assigned to every surface grid cell in the model domain, but horizontally-varying surface water temperatures may be put in the model by modifying subroutine SFCINITF_USER in the file **ruser.f** (see Section 9.1). Surface water temperature is relevant for any surface grid cell which is not completely covered by land (see description for PCTLCON), and strongly influences the fluxes of heat and moisture between the surface and atmosphere. The model assumes that the surface water temperature is constant in time during a simulation, as if the water is part of a large heat reservoir.

DTHCON is a parameter representing the temperature difference in degrees Kelvin obtained by subtracting a fictitious soil temperature from the temperature of the lowest level of the atmosphere. It is used as a simple means for imposing a surface source or sink of heat to the atmosphere in the case where the soil model is not activated, that is for ISFCL (see in this Section) set to 0. For example, setting DTHCON to -10 would impose a flux of heat into the atmosphere equivalent to that which would occur over a ground surface 10 K warmer than the air a height around one half DELTAZ (see in Section 7.2) above the ground.

DRTCON is similar to DTHCON, being relevant only for ISFCL (see in this Section) set to 0, but applies to a difference in water vapor mixing ratio, rather than to a temperature difference. For example, setting DTHCON to -0.002 would cause an upward flux of moisture to the atmosphere equivalent to that which would occur if the equilibrium vapor mixing ratio in the soil were 2 grams per kilogram higher than the ambient vapor mixing ratio at low levels in the atmosphere.

SOILDZ is a parameter used to specify the vertical spacing between successive levels of the soil model grid. The product of SOILDZ and NNZG (see in Section 7.2) minus 1 would give the total depth of the soil model. SOILDZ is not often used, because it is usually desirable for the soil model grid to be stretched vertically such that near the surface the spacing is around a few centimeters, and below, the spacing is on the order of 10 cm. This enhances the resolution near the surface where vertical temperature gradients are typically the strongest, while using coarser resolution below where sharp gradients do not often exist. The stretched grid option is activated by setting SOILDZ to 0, and specifying instead the actual depths below the surface of the individual soil levels in the namelist variable array SLZ (see in this Section). The soil model should normally extend to a depth around 1/2 meter for simulations lasting only a day or two, but should be a meter or even more for extended simulation periods.

SLZ is a multi-valued array of values, equal in number to NNZG (see in Section 7.2) denoting the number of levels in the soil model. SLZ specifies the depth in meters of each soil level beginning with the deepest level. For the SLZ values to be used, the namelist parameter SOILDZ (see in this Section) must be set to 0.

SLMSTR is a multi-valued parameter used to initialize the moisture content of the soil. It now applies both to simulations that are run with horizontally-homogeneous or with variable initialization (see description for INITIAL in Section 7.3). The number of values specified for it must equal the number of soil levels NNZG (see in Section 7.2), and the values must be separated by commas. The first value specified for SLMSTR applies to the deepest soil level, with the last value representing the topmost layer of soil. Allowable values for this parameter range from 0.0, representing totally dry soil, to 1.0 which represents totally saturated soil. A more realistic lower limit is about .15 since a totally dry soil can only be made by baking it! The values represent percentage of total water capacity which the soil can hold. The equilibrium water vapor mixing ratio of the soil is the quantity with which the ambient atmospheric water vapor mixing ratio interacts in determining the flux of moisture between the atmosphere and soil. This soil equilibrium value is a highly nonlinear function of the soil moisture content (represented initially by SLMSTR). The equilibrium vapor pressure is essentially constant for soil moisture below about 0.2 and above about 0.7 [depending on the soil type (see description for NSLCON in this Section)], and varies sharply in between those values. The SLMSTR values are by default used to fill all soil grid points horizontally homogeneously, but horizontal variations in initial soil moisture may be specified by the user by altering the code in subroutine SFCINIT_USER in subroutine **ruser.f** (see Section 9.1).

STGOFF is a multi-valued array of values, equal in number to NNZG (see in Section 7.2). STGOFF is used to define a vertical temperature profile in the initial conditions for the soil model, and represents a deviation from the temperature in degrees K in the lowest atmospheric level in the model. The values specified in the array must be separated by commas. The first value of STGOFF applies to the deepest level of soil, while the last value applies to the topmost soil level. For example, if the first value of STGOFF is set to 5.0, the deepest soil level will initially have a temperature 5 degrees K. warmer than the initial temperature of the lowest atmospheric level. As for many other soil variables in the namelist, the values specified in STGOFF are by default used to horizontally homogeneously initialize all soil model grid points. A user may impose horizontal variations in the initial soil temperature by modifying the code in subroutine SFCINIT_USER in module **ruser.f** (see Section 9.1).

IDIFFK is a multi-valued, grid-dependent flag which controls the type of parameterization to be used for computing both horizontal and vertical diffusion coefficients. Allowable values for this flag are 1, 2, 3, or 4. A value of 1 or 2 is appropriate to model grids in which the horizontal grid spacing is large compared to the vertical spacing such that dominant convective motions are not resolved. Horizontal diffusion in such cases is normally required to be stronger (for numerical damping) than justifiable on physical grounds, and the model assumes complete decoupling of the horizontal and vertical diffusion in all aspects, including computation of deformation rates, length scales, and stress tensor components. With IDIFFK set to either 1 or 2, the horizontal diffusion coefficients are computed as the product of horizontal deformation rate (horizontal gradients of horizontal velocity) and a length scale squared, based on the original Smagorinsky formulation. The length scale is the product of x-direction grid spacing DELTAX (see in Section 7.2) and namelist parameter CSX (see in this Section). When IDIFFK is set to 1, vertical diffusion is parameterized according to the Mellor and Yamada scheme, which employs a prognostic turbulent kinetic energy (TKE). When IDIFFK is set to 2, vertical diffusion is computed from a 1-dimensional analog of the Smagorinsky scheme in which vertical deformation is evaluated from vertical gradients of horizontal wind, and vertical length scale is the local vertical grid spacing times namelist parameter CSZ (see in this section). In addition, modifications of the vertical diffusion coefficient due to static stability or instability are used, based on formulations of Hill and Lilly. The Lilly modification is in the form of a multiplying factor, equal to sqrt(1-R kh/km) where R is the Richardson number and kh/km is the ratio of the scalar to momentum vertical diffusion coefficients specified by the user in namelist variable ZKHKM (see in this Section). The multiplying factor is greater than 1 in unstable cases (*i.e.*, where wind shear and/or unstable lapse rates make the Richardson number sufficiently low, and is less than 1 in stable cases. The Hill modification applies only to regions of unstable lapse rates (having negative squared Brunt-Vaisala frequencies), and consists of adding the absolute value of the Brunt-Vaisala frequency squared to the deformation rate, to obtain a modified inverse time scale for the diffusion coefficient computation. The Lilly and Hill modifications were originally designed for use without each other, although we have found that the added vertical diffusion in unstable air obtained by using both together is usually desirable. Values of 3 or 4 for IDIFFK are usually appropriate on grids having comparable horizontal and vertical spacings, and are intended for use in situations where dominant 3-dimensional motions, such as cumulus or boundary layer convection, are resolved. In these cases, RAMS computes diffusion coefficients from the 3-dimensional rate-of-strain tensor, and uses the same or similar diffusion coefficients for horizontal and vertical diffusion. When IDIFFK is set to 3, horizontal and vertical diffusion coefficients are computed as the product of the 3-D rate-of-strain tensor and a length scale squared. The length scale is the product of the vertical grid spacing and namelist parameter CSZ (described in this Section). The Hill and Lilly stability-dependent modifications described above also apply to IDIFFK set to 3. When IDIFFK is set to 4, vertical and horizontal diffusion are parameterized according to the Deardorff scheme, which employs a prognostic subgrid turbulent kinetic energy. This scheme is intended only for the specific purpose of performing a large eddy simulation (LES) in which it is assumed that resolved eddy motions in the model perform most of the eddy transport. The parameterized diffusion only represents the subgrid turbulent mixing. Thus, the Deardorff scheme is not appropriate for horizontal grid cell sizes larger than a couple hundred meters. In both the Deardorff and Mellor and Yamada schemes, the prognostic kinetic energy is generated by means of shear and buoyancy processes, and a parameterized pressure-work term, and is destroyed by potentially all of the above plus a parameterized dissipation term.

The kinetic energy is also advected and diffused. The result of these processes yields a field of kinetic energy from which the diffusion coefficients are locally diagnosed.

CSX is multi-valued, grid-dependent parameter serving as a coefficient in computing horizontal diffusion coefficients, in the case where IDIFFK (see in this Section) is set to either 1 or 2. It is used in computing the horizontal diffusive length scale, defined as the product of CSX and the x-direction grid spacing. Doug Lilly and others have proposed specific values for CSX based on theory, although the philosophy on which such a theory is based has been a matter of controversy. Setting CSX to 0.135 is effectively the value proposed by Lilly to yield diffusion coefficients of a strength that just provides adequate damping of the smallest resolvable features on the grid (to prevent an unrealistic accumulation of energy at that small scale). However, justification for a considerably larger of CSX of 0.32, has been argued by Paul Mason for more strongly filtering features represented on the grid, so that the only features with wavelengths longer than about 4 or 5 times the grid spacing will exist at any significant amplitude on the grid. This choice is desirable since the the smaller wavelengths of 2 and 3 times the grid spacing are not properly handled by numerical finite differencing schemes. The price paid for this stronger diffusive filter is the requirement of more grid cells for simulating a particular feature. The user must make a judgement here as to how strong to make CSX. This is often done on a case-by-case basis based on a subjective decision as to whether the model prognostic fields are "sufficiently smooth".

CSZ is similar to CSX (see in this Section), but it applies to the computation of vertical, rather than horizontal, diffusion coefficients, and is relevant only for IDIFFK (see in this Section) set to 2 or 3.

XKHKM is a multi-valued,grid-dependent parameter in which the user specifies the ratio of strength of horizontal diffusion coefficients between scalars and velocity to be used in a simulation. It is used to multiply the momentum diffusion coefficient to obtain the scalar coefficient, and to cases where IDIFFK (see in this Section) is set to 1 or 2. Deardorff found that the scalar coefficient had to be around 3 times larger than the momentum coefficient in strongly convective boundary layers for adequate mixing of all fields. For stable situations, the ratio may decrease to 1. It has been our usual practice to set XKHKM to 3, a value which applies throughout the model domain regardless of local stratification. The Deardorff parameterization for large eddy simulations (activated by setting IDIFFK to 4) includes an automatic computation of this ratio for each grid cell in the domain based on the local conditions.

ZKHKM is similar to XKHKM (see in this Section), but applies to the computation of vertical, rather than horizontal, diffusion coefficients. It is applicable only in cases where IDIFFK (see in this Section) is set to 2 or 3.

AKMIN is a multi-valued, grid-dependent parameter used to impose a minimum value on the value of the horizontal diffusion coefficients throughout the model domain. It is only relevant for cases where IDIFFK (see in this Section) is set to 1 or 2. The minimum value is sometimes required when the local fluid deformation rate happens to be close to zero in a region, resulting in very weak diffusion there. Setting AKMIN to a value greater than 0 will prevent the horizontal diffusion coefficients everywhere from falling below the minimum. AKMIN is scaled such that if set to 1.0, the minimum value placed on the diffusion coefficients should be about the same as normally obtained from the basic deformation scheme itself (without buoyancy effects) regardless of grid spacing. In practice, it has been found that AKMIN values up to 3.0 are sometimes required if boundary layer convection is strong (requiring higher damping than computational noise generated by shear alone). It is suggested that AKMIN be set to a low value, say

0.1, so that the standard horizontal diffusion parameterization be given a chance to do its job. If it is found that horizontal diffusion is inadequate (for example by considerable grid noise appearing in a contour plot of model fields), AKMIN may be increased, or CSX (see in this Section) may be increased, or both.

NLEVEL is a multi-valued, grid-dependent flag which specifies the level of moisture complexity to be activated in the model. This flag is the primary means by which the user tells the model whether or not to consider the effects of moisture in the simulation, and if so, to what degree. Setting NLEVEL to 0 causes the model to run dry, completely eliminating any process which influences or is influenced by any phase of moisture. With this option, radiation parameterization (controlled by ISWRTYP and ILWRTYP described in this Section) must be turned off. A value of 1 for NLEVEL activates advection, diffusion, and surface flux of water, where all water substance in the atmosphere is assumed to occur as vapor even if supersaturation occurs. The value of 1 also activates the buoyancy effect of water vapor in the vertical equation of motion, as well as the radiative effects of water vapor if radiation is activated elsewhere. NLEVEL equal to 2 activates condensation of water vapor to cloud water wherever supersaturation is attained. The partitioning of the total water substance into vapor and cloud water is always purely diagnostic. No other forms of liquid or ice water are considered. Both the positive buoyancy effect of water vapor and the liquid water loading of cloud water are included in the vertical equation of motion. Radiative effects of both water vapor and cloud water are activated, if the radiation parameterization is itself activated. Setting NLEVEL to 3 does everything that NLEVEL equal to 2 does, plus activating the microphysical parameterization of any species of liquid or ice specified in the microphysics flags described in this Section. This parameterization includes the precipitation process. **Note that in version 3b of RAMS, both the old and new microphysics packages are available. The selection between these is controlled by namelist variable ICLOUD, described below. However, use of the new microphysics rather than the old is strongly recommended, and the old microphysics package will no longer supported**.

ICLOUD is cloud parameter not yet fully functional in RAMS. However, in version 3b, setting ICLOUD to a negative value selects the old microphysics package, while setting it to a positive value selects the new microphysics package, subject, of course, to the setting of NLEVEL above.

IRAIN is a multi-valued, grid-dependent flag which controls the model's treatment of rain water. If set to 0, rain is not activated, and any process involving the interaction of rain with other water species is not performed. Rain is activated when IRAIN is set to a value from 1 to 5. Activation means that the mixing ratio of rain is prognosed from conservation equations which include advective, diffusive, and precipitation tendencies, and source terms resulting from interactions between rain and other forms of water substance. The choice of values for IRAIN between 1 and 5 controls the manner in which the mean rain droplet diameter and number concentration are determined. With the flag equal to 1, the mean diameter is specified from a default value in the code, and the number concentration is diagnosed automatically from this mean diameter and the prognosed mixing ratio. When the flag is set to 2, the user specifies the mean diameter (in meters) in the namelist variable RPARM, and the number concentration is diagnosed as before. When the flag is set to 3, the user specifies the y-intercept value of the number concentration (this is the number concentration per unit diameter increment, (*i.e.*, number per m^3 per m), evaluated at zero diameter, assuming a Marshall-Palmer (exponential) size distribution. The total number concentration (*i.e.*, the total number of droplets per cubic meter) and the mean diameter are then both diagnosed from the y-intercept value and the prognosed mixing ratio. With a value of 4 for the flag, the user specifies the total number concentration of rain droplets (number per m^3) directly in the namelist variable RPARM, and the mean diameter is automatically diagnosed from this concentration and the prognosed mixing ratio. When the flag is set to 5, the model activates a prognostic equation for rain droplet number concentration, and a special array for this quantity is added. Mean rain droplet diameter is then diagnosed from the prognosed mixing ratio and number concentration. AT PRESENT, THIS OPTION OF PROGNOSING NUMBER CONCENTRATION IS NOT IMPLEMENTED FOR RAIN (NOR FOR SNOW, AGGREGATES, GRAUPEL, AND HAIL) BUT ONLY FOR PRISTINE ICE. The choice of which value of IRAIN to use will depend to a large extent on how the user wishes to constrain the rain field, *i.e.*, whether mean diameter, total number concentration, or both should be allowed to vary. For example, if the user is simulating an observed and documented precipitation event in which, say, mean droplet diameter was measured, it would make sense to specify that diameter as a fixed quantity. Independent selection of microphysical species to be activated versus not activated is controlled through the setting of IRAIN and the related parameters IPRIS, ISNOW, IAGGR, IGRAUP, and IHAIL described below. However, not all possible combinations are intended as valid nor would they function properly without code modification. Activation of all categories simultaneously is the normal practice, although activation of cloud only, cloud and rain only, cloud and pristine ice only, or pristine ice only are possible when only a limited range of microphysical processes need to be considered.

IPRIS is similar to IRAIN, but controls the activation of pristine ice rather than rain. The descriptions of the various settings given for IRAIN apply here, except that the prognostic concentration option (IPRIS equal to 5) is functional.

ISNOW is similar to IRAIN, but controls the activation of snow rather than rain. The descriptions of the various settings given for IRAIN apply here.

IAGGR is similar to IRAIN, but controls the activation of aggregates rather than rain. The descriptions of the various settings given for IRAIN apply here.

IGRAUP is similar to IRAIN, but controls the activation of graupel rather than rain. The descriptions of the various settings given for IRAIN apply here.

IHAIL is similar to IRAIN, but controls the activation of hail rather than rain. The descriptions of the various settings given for IRAIN apply here. IHAIL is only functional in the new microphysics version.

CPARM is similar to RPARM below, but applies to cloud water rather than rain. It is only functional for the new microphysics package.

RPARM is a variable in which the user specifies either a fixed mean diameter, a fixed total number concentration, or a fixed y-intercept number concentration per unit diameter increment to be imposed on the rain droplet field, in conjunction with the setting for the flag IRAIN described above. Values for any are in mks units.

PPARM is similar to RPARM, but applies to pristine ice rather than rain.

SPARM is similar to RPARM, but applies to snow rather than rain.

APARM is similar to RPARM, but applies to aggregates rather than rain.

GPARM is similar to RPARM, but applies to graupel rather than rain.

HPARM is similar to RPARM, but applies to hail rather than rain, and will be functional only in the future microphysics package currently being tested.

AMIO is a parameter in which the user specifies the minimum mass in *kg* of pristine ice crystals. The parameter is relevant only if microphysical parameterization is activated in the model, and if as part of this parameterization, pristine ice is activated by setting IPRIS (see in this Section) greater than 0. AMIO is used in the model nucleation schemes which convert an amount of water vapor into pristine ice based on appropriate nucleation formulas, and denotes the initial mass an ice crystal has upon nucleation; further diffusional growth proceeds from this initial mass. A value of 1.E-12 is suggested for AMIO unless the user has better information relevant to the simulation being run.

GNU is a parameter with 7 values, one for each hydrometeor category. It specifies the shape parameter of the gamma distribution which all categories (**in the new microphysics package**) are assumed to follow. GNU equal to 1 indicates the Marshall-Palmer, or exponential, distribution, in which number concentration decreases monotonically with diameter throughout the size spectrum. Larger values of GNU indicate more general gamma distributions, in which the size distribution peaks at a positive diameter. The larger the value of GNU, the more narrowly distributed the spectrum is. The intent in using this variable is to specify a shape based on observation. The shape may in general depend on the type of precipitating system (deep convective, wintertime cyclone, etc), as well as on the hydrometeor category (rain versus hail, for example). Very little observational guidance is available to date on appropriate values of GNU. A Marshall Palmer distribution has been used in many models, even though it may not be the closest of the gamma distribution family to real hydrometeor spectra. Sensitivity experiments with GNU in RAMS have demonstrated that it can sometimes have a significant effect on model results. The user is encouraged to experiment with the values of GNU, and to use observational guidance where possible. Ongoing research is aimed at determining appropriate values of GNU for each hydrometeor category in different weather situations.

Section 7.5 - The $MODEL_SOUND Namelist

The **$MODEL_SOUND** namelist consists of a set of variables for specifying a sounding to be used in initializing a simulation. This method of initialization is performed horizontally homogeneously, and is performed only when the flag INITIAL (see in Section 7.3) is set to 1. The variables in the **$MODEL_SOUND** namelist consist mainly of (1) arrays containing the actual values of pressure or height, velocity, temperature, and moisture, and (2) a set of flags specifying how the array values are to be interpreted. An alternative form of specifying five of the variables in this namelist, PS, TS, RTS, US, and VS, all described below, is to enter them into a file called **SOUND_IN** in tabular form. This option is selected by setting the first value of PS (see below) to 0. In this case, the model opens and reads the file **SOUND_IN**, which must reside in the same directory as the model executable. The code that reads from this file is contained in the file **rhhi.f** in subroutine ARRSND in the first Do Loop. Free format is assumed with one sounding level per record starting with the lowest sounding level at the top of the file, and the five variables are read from each record in the order shown above.

IPSFLG is a flag specifying how the values given for PS (see in this Section) are to be interpreted. When IPSFLG is set to 0, all values in PS are interpreted as pressures in millibars, and when IPSFLG is set to 1, all values in PS are interpreted as heights in meters. In either case, the PS values are used to define the heights in the sounding at which all sounding data applies.

ITSFLG is a flag specifying how the values given for TS (see in this Section) are to be interpreted. When ITSFLG is set to 0, all values in TS are interpreted as temperatures in degrees C, when ITSFLG is set to 1, all values in TS are interpreted as temperatures in degrees K, and when ITSFLG is set to 2, all values in TS are interpreted as potential temperatures in degrees K. The options allow the model to read sounding temperature data in any of these forms.

IRTSFLG is a flag specifying how the values given for RTS (see in this Section) are to be interpreted. When IRTSFLG is set to 0, all values in RTS are interpreted as dew point temperatures in degrees C, when IRTSFLG is set to 1, all values in RTS are interpreted as dew point temperatures in degrees K, when IRTSFLG is set to 2, all values in RTS are interpreted as water mixing ratio values in grams per kilogram, when IRTSFLG is set to 3, all values in RTS are interpreted as relative humidities in percentage, and if IRTSFLG is set to 5, all values in RTS are interpreted as dew point depressions in degrees K. The options allow the model to read sounding moisture data in any of these forms.

IUSFLG is a flag specifying how the values given for US and VS or for USNDG and VSNDG (see in this Section) are to be interpreted. When IUSFLG is set to 0, all values in US (or USNDG) are interpreted as components of velocity in meters per second in the x-direction, and all values in VS (or VSNDG) are interpreted as components of velocity in meters per second in the y-direction. When IUSFLG is set to 1, all values in US (or USNDG) are interpreted as wind directions in degrees azimuth clockwise from true north, and all values in VS (or VSNDG) are interpreted as wind speeds in meters per second in the x-direction.

IUSRC is a flag specifying the source of wind data to be used in initializing the model. The choice is between two pairs of multi-valued variables in the namelist: US with VS, or USNDG with VSNDG. If IUSRC is set to 0, the winds specified in US and VS (subject to the interpretation specified in IUSFLG (see in this Section)) are interpreted from their associated sounding levels (as specified in PS (see in this Section)). IF IUSRC is set to 1, the wind information specified in USNDG and VSNDG (again subject to the interpretation specified in IUSFLG (see in this Section)) are assigned directly to the model levels without vertical interpolation. This implies that the number of values specified for USNDG and VSNDG must equal NNZP (see in Section 7.2) for the coarse grid.

HS is a multi-valued array used to define sounding heights, but only the first value is read from the namelist. This first value indicates the absolute height (relative to sea level) of the first or lowest sounding level. It must be at or below the lowest topography height in the RAMS model domain.

PS is a multi-valued array whose values represent either the pressures in millibars or the heights in meters of the sounding levels starting at the ground and proceeding upward. The choice of meaning is controlled by the value of IPSFLG (see in this Section). If PS represents heights, the first value of PS is still specified as a pressure in millibars, the height of which is read from namelist variable HS (see in this Section). The primary requirement of the sounding data is that it extend to a height greater than the top of the model domain, and at least as low as the lowest topography in the domain. Thus, the top sounding level must contain either a height or a pressure higher in the atmosphere than the model domain top, which is itself determined by the combination of namelist parameters NNZP, DELTAZ, DZRAT, and DZMAX, or by ZZ (see in Section 7.2). A maximum of 200 values may be specified for PS, although with code modification this number could be increased if necessary. The values given for PS will determine the vertical spatial resolution of the sounding, and should be chosen to properly represent any important significant levels. **Note the special option of setting the first value of PS to 0., as described at the beginning of this Section.**

TS is a multi-valued array whose values represent sounding temperatures. The precise way in which the values are interpreted is specified by namelist variable ITSFLG (see in this Section). The number of values specified for TS must equal the number of values specified for PS (see in this Section). The user should use caution in specifying sounding temperatures. Superadiabatic profiles in the sounding are strongly discouraged (which does not mean that the model simulation may not successfully develop superadiabatic regions after the initialization).

High vertical resolution is sometimes desirable in a sounding because the model fields of temperature and moisture mixing ratio are interpolated linearly from it in the vertical direction, and relative humidity is a very nonlinear function of these two interpolated fields. This can result in supersaturated layers in the model initial fields where none exist in the sounding.

RTS is a multi-valued array whose values represent sounding moisture values. The precise way in which the values are interpreted is specified by namelist variable IRTSFLG (see in this Section). The number of values specified for RTS must equal the number of values specified for PS (see in this Section). (See also the description given for TS in this Section).

US is a multi-valued array whose values represent sounding winds. The precise way in which the values are interpreted is specified by namelist variable IUSFLG (see in this Section). The number of values specified for US must equal the number of values specified for PS (see in this Section).

VS is a multi-valued array whose values represent sounding winds. The precise way in which the values are interpreted is specified by namelist variable IUSFLG (see in this Section). The number of values specified for VS must equal the number of values specified for PS (see in this Section).

USNDG is a multi-valued array whose values represent initial winds to be assigned directly to the model grid levels. The precise way in which the values are interpreted is specified by namelist variable IUSFLG (see in this Section), and the choice of whether to use USNDG (along with VSNDG) as a data source, as opposed to using US (along with VS), is controlled by namelist variable IUSRC (see in this Section). The number of values specified for USNDG must equal NNZP (see in Section 7.2) for the coarse grid.

VSNDG is a multi-valued array whose values represent initial winds to be assigned directly to the model grid levels. The precise way in which the values are interpreted is specified by namelist variable IUSFLG (see in this Section), and the choice of whether to use VSNDG (along with USNDG) as a data source, as opposed to using VS (along with US), is controlled by namelist variable IUSRC (see in this Section). The number of values specified for VSNDG must equal NNZP (see in Section 7.2) for the coarse grid.

KMEAN1 is a parameter in which the user specifies the vertical level (k index) in the model above which a vertical average of the initial horizontal wind will be taken for use in a Galilean transformation. The upper limit of the vertical average is specified in KMEAN2 (see in this Section). The wind averaged between these two values is placed into variables UMEAN and VMEAN (see their descriptions as namelist variables).

KMEAN2 is a parameter in which the user specifies the vertical level (k index) in the model below which a vertical average of the initial horizontal wind will be taken for use in a Galilean transformation. The lower limit of the vertical average is specified in KMEAN1 (see in this Section). The wind averaged between these two values is placed into variables UMEAN and VMEAN (see their descriptions as namelist variables). If KMEAN2 is set to 0, this vertical average of winds will not be performed.

UMEAN is a parameter in which the user may specify a component of motion in meters per second in the x-direction for the purpose of performing a Galilean transformation on the model solution. This velocity component is subtracted uniformly from every velocity value in the model domain, which has the effect of subtracting the x-direction motion of all meteorological disturbances by that amount. (The value of UMEAN is accounted for in the surface flux calculations of heat, moisture, and momentum so they are not altered.) The Galilean transformation is used to keep disturbances of importance within the model domain, or within a particular nested fine grid, for the duration of a simulation, where they would otherwise propagate or be advected outward too soon. This option is not appropriate unless the ground surface in the model is featureless. If variations in surface features, such as terrain height, are present in the simulation, the user has either the option of making a grid large enough to contain the traveling disturbance within the duration of the simulation, or of moving a grid in time if the grid is a nested fine mesh (for this option see the description for GRIDU). An alternate option for specifying UMEAN is to instruct the model to fill it from a vertical average of the initial winds over a layer defined by KMEAN1 and KMEAN2 (see in this Section).

VMEAN is similar to UMEAN, but applies to the y-component, rather than the x-component, of motion used in the Galilean transformation.

Section 7.6 - The $MODEL_PRINT Namelist

The **$MODEL_PRINT** namelist provides a means for obtaining a quick look at model fields. It is used to specify selected data from the model to be written to the standard output file generated with a model run. This data is then examined by displaying the contents of the standard output file.

NPLT is a parameter specifying the total number of data subsets to be written to the standard output file. Each of these data subsets consists of a 2-D cross section of values from a 3-D model field. Different cross sections from the same model field are considered as separate data subsets. (If the model itself is run in 2-D, the 2-D cross section normally encompasses the entire model domain.) The exact specification of the cross section to be output is made in namelist variables IPLFLD, IXSCTN, and ISBVAL (see in this Section).

IPLFLD is a multi-valued character variable whose values denote the field variables to be written to the standard output file. The values specified for IPLFLD must each be followed by a comma, and the total number of values specified must be at least the value specified for NPLT (see in this Section). The authentic complete list of character strings which can be used as values for IPLFLD is provided in FUNCTION OPTLIB located in the **rprnt.f** module.

PLFMT is a multi-valued character variable whose values denote the FORTRAN format to be used in writing a data subset (see NPLT) to the standard output file. Although a default format exists in the model for every field variable, defining alternative formats in PLFMT is a way of overriding the default when necessary. Any element of PLFMT which the user wishes to define a value for is listed individually. We illustrate the use of this variable by means of the following example: PLFMT(2)='3PF6.1'. The subscript 2, denoting the second element in the PLFMT array, corresponds to the second element specified in the IPLFLD array (see in this Section). The character string assigned to PLFMT(2) denotes the format to be used in writing that second field in IPLFLD. The format statement itself contains the F6.1 part referring to the well-known floating point format, and the less-well-known 3P part which scales (multiplies) each value by 10**3 before writing it. This latter feature is sometimes necessary when values in a given field become unusually large or small, and are thus incompatible with the default format. For example, the default format for vertical velocity involves a multiplication of the model values by 100 so that output values are expressed in cm/sec. This is usually best for mesoscale simulations, but not appropriate for simulations of deep convection which generate vertical velocities of tens of meters per second.

IPLTYP is a multi-valued character variable flag for which a number of values equal to or greater than NPLT (see in this Section) must be specified. IPLTYP does not currently control a choice of option, and should just be set to 'PR' for each data subset plotted. Each of these character strings must be followed by a comma.

IXSCTN is a multi-valued flag for which a number of values equal to or greater than NPLT (see in this Section) must be specified. Each element in the IXSCTN array refers to the corresponding element in the IPLFLD and ISBVAL arrays (see in this Section). IXSCTN controls the orientation of the 2-D cross section extracted from the given model field for writing to the standard output file. A value of 1 for IXSCTN specifies the data subset to be an X-Z cross section, a value of 2 denotes a Y-Z cross section, and a value of 3 denotes an X-Y cross section.

ISBVAL is a multi-valued flag for which a number of values equal to or greater than NPLT (see in this Section) must be specified. Each element in the IXSCTN array refers to the corresponding element in the IPLFLD and IXSCTN arrays (see in this Section). IXSCTN selects a specific 2-D slab from a 3-D model field among all slabs having the orientation specified by IXSCTN. Each number specified for IXSCTN refers to the grid location index in the direction perpendicular to the slab, with the value 1 denoting either the westernmost, the southernmost, or the lowest slab, depending on the orientation.

CHAPTER 8 - GRAPHICAL DEPICTION OF RAMS MODEL OUTPUT: THE VAN AND REVU PACKAGES

This chapter describes VAN and REVU which are the standard supported packages for generating graphical representations of RAMS model output. Their function is to read *analysis files* written from a RAMS run, select user-specified fields and cross sections from the file data, and plot the field cross sections. Both VAN and REVU utilize NCAR graphics to perform most of the plot functions.

It is not difficult to interface RAMS model results to other graphics packages, particularly if one is familiar with a particular package. One option is for the user to insert write statements at the appropriate place(s) in the RAMS model code to output desired fields at desired times to a file in a format of one's own choosing. Matching read statements are added to graphics package, possibly along with new array space, making those fields available for plotting. Another option is to use the "DUMP" facility in REVU. It would also be possible to read the standard RAMS analysis files into another graphics package, although some RAMS code is required to unpack the data..

REVU is a newer, more efficient package that will replace VAN. Both are supported in this version of RAMS. An important advantage to REVU is that whereas VAN must be compiled with exactly the same include files (**rconfig.h** amd **rcommons.h**) as the RAMS model, REVU has few ties to the model code and obtains all the model information it needs entirely from the analysis files. Another advantage is that VAN always reads the entire analysis file regardless of what fields will be plotted while REVU reads only fields selected for plotting. REVU is still under development and lacks some of its future plotting capabilities. These will include contouring variable fields with colorfill or grey shades and plotting variables as a function of time. Current capabilities of REVU that are not available in VAN are plotting maps in colorfill, plotting wind barbs, and color coding plotted velocity vectors according to scalar values of a selected field.

The processes of setting namelist parameters and running VAN and REVU are quite similar even though their codes are completely different. The normal procedure is to run either of the executables in the rams3b/bin subdirectory, RAMS_van or Revu, or if changes must be made to the code, to copy the archive RAMS_van.a or Revu.a, the makefile Make.van or Make.revu, and the code file(s) requiring changes to the working directory. In this latter case, make the code changes and compile the changed file(s) following the procedure described in Chapter 4. Also copy the namelist file VAN_IN or REVU_IN to the working directory, set namelist variables appropriately, and run the executable. Running VAN or REVU produces a NCAR Graphics metacode file which can be viewed or printed using the metacode translators.

The remainder of this chapter describes the namelist parameters that control plotting options in VAN and REVU.

Section 8.1 - VAN Namelist Variables

The namelist file VAN_IN contains two namelists, with the names **$CONTROL** and **$FIGURES**. The **$CONTROL** namelist specifies the model analysis file names and what type of run to do. The **$FIGURES** namelist contains the parameters necessary to make plots with flexible orientations, including time plots. It also contains many optional parameters used to label the plot and change the behavior of the plotting routines. Note that an index of configuration and namelist variables for chapters 6, 7, and 8 is located at the end of chapter 8.

Section 8.1.1 - The $CONTROL Namelist

ANPREF is the filename "prefix" of the input analysis files including any required absolute or relative path. This is the same prefix (**AFILOUT**) specified in the RAMS model namelist file RAMSIN (see Section 7.3) for the output analysis files except that the path may of course be different. VAN finds all files in the specified directory that have this prefix and determines their chronological order from the time suffix encoded into the filenames. The user selects some or all of these files to be plotted in the namelist variable **TVAR** (see below).

ANATYPE Designates what type of run to make. It can be set to: 'SPACE' - All plots requested will only have spatial axes. 'TIME' - All plots requested will have a time axis.

Section 8.1.2 - The $FIGURES Namelist

HEAD1 is a 24 character long string used to title all plots.

HEAD2 is another 24 character string written below the one from HEAD1 on every plot.

NBACKS is an integer specifying how many backgrounds will be plotted. A background is a particular choice of two dimensional slab with a defined size, location, orientation, and grid number(s). By default, there can be up to 3 different backgrounds plotted in a single run. For each type of background several foregrounds or individual plots of model fields may be generated.

IGRID specifies which grid is to be plotted. If it is set to zero, all grids will be plotted (on separate frames). The subscript of IGRID refers to the background number (see NBACKS above), and a separate IGRID value must be specified for each background.

IZTRAN If this background is a horizontal cross-section, the foregrounds can either be plotted along the model's terrain-following coordinate surface (IZTRAN=0), on a horizontal surface (IZTRAN=1), or on a constant pressure surface (IZTRAN=2). For this third option, the surface is specified in IPLEVEL (see below). The subscript of IZTRAN refers to the background number (see NBACKS above), and a separate IZTRAN value must be specified for each background.

IPLEVEL If this background is a horizontal cross-section and IZTRAN is set to 2 (see above), IPLEVEL denotes a pressure surface (in mb) on which to generate the plot. The subscript of IPLEVEL refers to the background number (see NBACKS above), and a separate IPLEVEL value must be specified for each background.

CMAP specifies whether a map projection will be plotted. It is relevant only if this plot has X plotted horizontally and Y plotted vertically (see XVAR and YVAR below). It will take the values of 'yes' or 'no'. If this parameter is set to a space, a normal graph background is drawn. The subscript of CMAP refers to the background number (see NBACKS above), and a separate CMAP value must be specified for each background.

XVAR, YVAR, ZVAR, and **TVAR** are character strings that specify the orientation, location, and size of the two-dimensional slab to be plotted. The subscript of each variable refers to the background number (see NBACKS above), and a separate value for them must be specified for each background. The character strings specified for these variables are cryptic but are easier to use than specifying 20 different variables. Currently, there are two parts of the string that are delimited by the forward slash character /. The first part of the string is a single character which describes how this direction will appear on the background. The characters and their meanings are:

H - direction will be plotted horizontally.

V - direction will be plotted vertically.

F - fix the direction to the following value.

M - take a mean along this direction.

For example:

```
TVAR(1) = '/F/13:13:1/',
ZVAR(1) = '/V/0:0/',
YVAR(1) = '/F/1:1/',
XVAR(1) = '/H/0:0/',
```

This means that this background will have X horizontally on the plots and Z vertical. The Y direction will be fixed as will time T.

```
TVAR(1) = '/F/1:1:1/',
ZVAR(1) = '/F/4:4/',
YVAR(1) = '/F/1:1/',
XVAR(1) = '/H/0:0/',
```

This example has a line graph of the variables that will be plotted on this background. The X direction will be the abscissa while all other directions are fixed at some value. The ordinate of the plot will be the value of the variable being plotted.

```
TVAR(1) = '/F/1:1:1/',
ZVAR(1) = '/V/1:20/',
YVAR(1) = '/M/10:20/',
XVAR(1) = '/H/0:0/',
```

This example has the X direction along the abscissa, the Z direction along the ordinate, and a mean taken in the Y direction.

The second part of the character string specifies the range information for the directions. The syntax is left_value : right_value : increment , where the values, if positive, are actual grid point values. If the left or right values are negative or zero, they are interpreted as an offset from the boundaries of the grid. For the time direction, the values refer to the analysis file number. The increment will be ignored for the H,V, or F (other than TVAR, see below) specifications. If the increment is not specified, it defaults to 1.

So, for another example, XVAR(1)='/H/-2:-3/' means a horizontal axis starting at grid point 3 on the left (an offset of 2 from grid point 1) and ending at three gridpoints in from the right boundary.

TVAR is a special case. Often, one wants to plot several different times with the same background. If TVAR is fixed, a third parameter in the second part of the string can be specified. This is an increment through the analysis files. Example : TVAR(1)='/F/1:13:4/' means to plot this background at analysis files 1, 5, 9, and 13. These file numbers refer to the chronological order of all files in the specified directory of the specified prefix (see ANPREF above).

If a direction is set to M for a mean value, the second part of the string will be similar to TVAR. For example, if YVAR was '/M/0:0:2/', then an average would be taken in the Y direction for the entire domain at every other point. By the way, the mean will be a terrain following average along the sigma-z coordinate surfaces.

XAXLAB is an 8-character string used to label the horizontal axis. Reasonable defaults have been implemented, so this is optional. The subscript of XAXLAB refers to the background number (see NBACKS above), and a separate XAXLAB value may be specified for each background.

YAXLAB is like XAXLAB but is used to label the vertical axis.

LABFMTX is a string containing Fortran format descriptors without the enclosing parenthesis. It is used to format the numeric labels along the X axis. It is defaulted so the spatial variables will be plotted in kilometers. Examples: 'F6.0','-3PF7.2'. The subscript of LABFMTX refers to the background number (see NBACKS above), and a separate LABFMTX value may be specified for each background.

LABFMTY is like LABFMTX but pertains to the Y axis.

NUMLABX is an integer specifying approximately how many labels you wish to plot on the horizontal axis. Reasonable defaults have been implemented, so this is optional. The subscript of NUMLABX refers to the background number (see NBACKS above), and a separate NUMLABX value may be specified for each background.

NUMLABY is like NUMLABX but pertains to the vertical axis.

NPLOTS is an integer specifying how many foregrounds will be plotted on this background. Each foreground is a separate plot and may consist of up to one contoured field, one vector field, one streamline field, and a map plot, or any one, two, or three of the above. By default, up to 20 foregrounds can be plotted with a given background. (Increasing this limit is done by increasing the parameter MAXFORE in the file **vcommons.h** and recompiling the entire VAN code (all source code files located in the /van directory.) The subscript of NPLOTS refers to the background number (see NBACKS above), and a separate NPLOTS value may be specified for each background.

CCONTUR specifies which variable is to be contoured (by the NCAR graphics routine CONREC) on this plot (foreground). Possible choices are described below. Set to space (' ') if there should be no contour plot in this foreground. The first subscript of CCONTUR refers to the background number and the second subscript to the foreground number.

CONRLO is the minimum contour level to be used in CONREC. A value of zero allows the routine to choose its own minimum. The subscripts of CONRLO denote the background and foreground and correspond to those of CCONTUR.

CONRHI is the maximum contour level to be used in CONREC. A value of zero allows the routine to choose its own maximum. The subscripts of CONRHI denote the background and foreground and correspond to those of CCONTUR.

CONRINC if positive, it is the contour interval. If it is set to a negative number, then it indicates about how many contour lines will be drawn between the lowest and highest values. If set to zero, CONREC will choose the value. The subscripts of CONRINC denote the background and foreground and correspond to those of CCONTUR.

CVECTOR specifies if velocity vectors should be included on this foreground. Set to 'YES' if this is desired. The subscripts of CVECTOR denote the background and foreground and correspond to those of CCONTUR.

CSTRVAR specifies if a streamline field should be included on this foreground. Set to 'YES' if this is desired. The subscripts of CSTRVAR denote the background and foreground and correspond to those of CCONTUR.

These variables may be chosen for CCONTUR :

U u-component

V	v-component
W	w-component
GEO	geopotential height
PI	perturbation Exner function
PI0	reference state Exner function
THETA	potential temperature
TG	bare soil temperature
TGS	shaded soil temperature
SLM	bare soil moisture
SLMS	shaded soil moisture
TKE	prognostic turbulent kinetic energy
TEMP	air temperature
RTOTAL	total water mixing ratio
RVAP	water vapor mixing ratio
RCLOUD	mixing ratio of cloud water
RELHUM	relative humidity
RCOND	mixing ratio of all condensate
RRAIN	mixing ratio of rain
RPRISTIN	mixing ratio of pristine ice
RSNOW	mixing ratio of snow
RAGGREG	mixing ratio of aggregates
RGRAUP	mixing ratio of graupel
RHAIL	mixing ratio of hail
RICE	mixing ratio of all ice condensate
CRAIN	concentration of rain
CPRISTIN	concentration of pristine ice
CSNOW	concentration of snow
CGRAUP	concentration of graupel
CAGGREG	concentration of aggregates
CONPCP	convective precipitation rate (from parameterization)

CONACC accumulated convective precipitation (from parameterization)

TOPO topography height

PCTL land percentage

PRAIN accumulated precipitation of rain

PSNOW accumulated precipitation of snow

PAGGREG accumulated precipitation of aggregates

PGRAUP accumulated precipitation of graupel

PHAIL accumulated precipitation of graupel

PTOTAL accumulated total precipitation (resolved)

SPEED wind speed

RELVORTX x-component of relative vorticity

RELVORTY y-component of relative vorticity

RELVORTZ z-component of relative vorticity

TOTVORTZ z-component of absolute vorticity

MSLP reduced mean sea level pressure

Section 8.2 - REVU Namelist Variables

Some namelist variables in the REVU namelist file REVU_IN are identical both in name and function with those in the VAN namelist file VAN_IN and are repeated here for completeness. Others have certain differences. Unlike VAN, REVU does not define multiple "backgrounds" in a single run. Only one namelist called **$CONTROL** exists in REVU_IN. The namelist variables are described below.

ANPREF is the filename "prefix" of the input analysis files including any required absolute or relative path. This is the same prefix (**AFILOUT**) specified in the RAMS model namelist file RAMSIN (see Section 7.3) for the output analysis files except that the path may of course be different. REVU finds all files in the specified directory that have this prefix and determines their chronological order from the time suffix encoded into the filenames. The user selects some or all of these files to be plotted in the namelist variable **TVAR** (see below).

ANATYPE Designates what type of run to make. If it is set to 'SPACE', fields selected in CFRAME (see below) will be plotted with only spatial axes. (Time axis plots will be available in the future.) If ANATYPE is set to 'DUMP', fields selected in CFRAME are not plotted but are passed to a subroutine in file **dumpout.f** in the rams3b/mrc/revu directory. This file must be modified by the user to define the data structure format of this output. The purpose of this option is to extract fields from analysis files which are written in packed format. If ANATYPE is set to 'GRAB', model fields are not plotted but are interpolated to arbitrary points in space and time (not yet a functional option).

HEAD1 is a 24 character long string used to title all plots.

HEAD2 is another 24 character string written below the one from HEAD1 on every plot.

IGRID specifies which grid is to be plotted. If it is set to zero, all grids will be plotted (on separate frames).

IZTRAN If plotting a horizontal cross-section (see ZVAR, XVAR, and YVAR below), fields can either be plotted along the model's terrain-following coordinate surface (IZTRAN=1), on a horizontal surface (IZTRAN=2), or on a constant pressure surface (IZTRAN=3). For this third option, the surface is specified in IPLEVEL (see below).

IPLEVEL If plotting a horizontal cross-section (see ZVAR, XVAR, and YVAR below) and IZTRAN is set to 3 (see above), IPLEVEL denotes a pressure surface (in mb) on which to generate the plot. Only standard pressure levels 1000, 900, 850, 700, 600, 500, 400, 300, 200, and 100 mb may be used.

MAPFILL specifies whether a map projection will be plotted and if so, whether it will be color-filled. It is relevant only if this plot has X plotted horizontally and Y plotted vertically (see XVAR and YVAR below). It will take the values -1 indicating not to draw a map, 0 indicating to draw a non-color-filled map, and 1 indicating to draw a color-filled map.

XVAR, YVAR, ZVAR, and **TVAR** are character strings that specify the orientation, location, and size of the two-dimensional slab to be plotted. The character strings specified for these variables are cryptic but are easier to use than specifying 20 different variables. Currently, there are two parts of the string that are delimited by the forward slash character /. The first part of the string is a single character which describes how this direction will appear on the background. The characters and their meanings are described below. Note however that the option of plotting variables with respect to time is not yet implemented.

H - direction will be plotted horizontally.

V - direction will be plotted vertically.

F - fix the direction to the following value.

For example:

```
TVAR = '/F/13:13:1/',
ZVAR = '/V/0:0/',
YVAR = '/F/1:1/',
XVAR = '/H/0:0/',
```

This means that this background will have X horizontally on the plots and Z vertical. The Y direction will be fixed as will time T.

```
TVAR = '/F/1:1:1/',
ZVAR = '/F/4:4/',
YVAR = '/F/1:1/',
XVAR = '/H/0:0/',
```

This example has a line graph of the variables that will be plotted on this background. The X direction will be the abscissa while all other directions are fixed at some value. The ordinate of the plot will be the value of the variable being plotted.

The second part of the character string specifies the range information for the directions. The syntax is left_value : right_value : increment , where the values, if positive, are actual grid point values. If the left or right values are negative or zero, they are interpreted as an offset from the boundaries of the grid. For the time direction, the values refer to the analysis file number. The increment will be ignored for the H,V, or F (other than TVAR, see below) specifications. If the increment is not specified, it defaults to 1.

So, for another example, XVAR(1)='/H/-2:-3/' means a horizontal axis starting at grid point 3 on the left (an offset of 2 from grid point 1) and ending at three gridpoints in from the right boundary.

TVAR is a special case. Often, one wants to plot several different times with the same background. If TVAR is fixed, a third parameter in the second part of the string can be specified. This is an increment through the analysis files. Example : TVAR(1)='/F/1:13:4/' means to plot this background at analysis files 1, 5, 9, and 13. These file numbers refer to the chronological order of all files in the specified directory of the specified prefix (see ANPREF above).

CFRAME performs the combined functions of VAN namelist variables NPLOTS, CCONTUR, CONRLO, CONRHI, CONRINC, CVECTOR, and CSTRVAR plus a few additional functions. The subscript of CFRAME denotes frame number, and the number of CFRAME values defined in the namelist indicates the number of frames to plot (for a given grid). A single lengthy character string is assigned to CFRAME for each subscript value and indicates which fields are to be plotted in the frame and how they are to be plotted The string begins and ends with a single slash (/) character and contains 11 additional slash characters in between; these 13 slash characters are delineators for 12 individual information slots in the string. The information in each is specified by slot number below:

1. Character variable indicating the field to be contoured. Possible choices are listed below. Set to "none" if there should be no contour plot in this foreground.
2. Minimum contour level (for CONREC) for the field named in slot 1. A value of zero allows the routine to choose its own minimum.
3. Maximum contour level (for CONREC) for the field named in slot 1. A value of zero allows the routine to choose its own maximum.
4. If positive, this is the contour interval (for CONREC) for the field named in slot 1. If negative, then it indicates about how many contour lines will be drawn between the lowest and highest values. If set to zero, CONREC will choose the value.
5. Character value to denote how to depict the wind field. 'vector' indicates to plot the velocity field with vectors, 'barbs' to plot the velocity field with wind barbs, 'streamline' to plot the streamline field, or 'none' indicating not to plot velocity. Each value can be abreviated to its first character.
6. Integer variable indicating the number of grid points between adjacent wind vector or barb plots.
7. Specified vector length in grid point distance units of the vector velocity specified in slot 8.
8. Specified vector velocity corresponding to the vector length specified in slot 7.
9. Character variable indicating the second field to be plotted. Currently the only to plot this second field is by color coding of the vectors, if they are plotted (as instructed in slot 5).
10. Minimum contour level for the field named in slot 9.
11. Maximum contour level for the field named in slot 9.
12. If negative, this is the number of colors that will be used between the lowest and highest values of the field named in slot 9. If set to zero, a default number of colors will be used.

The following variables are currently implemented for plotting and may be specified for CFRAME. Adding new variables is straightforward and is done in the file **hvlib.f** in the **post** subdirectory. Obviously, the appropriate RAMS options must be activated to view many of these.

u	u-component of velocity on polar stereographic grid
v	v-component of velocity on polar stereographic grid
ue	westerly velocity component in earth coordinates
ve	southerly velocity component in earth coordinates
w	w-component of velocity
geo	geopotential height
speed	horizontal wind speed
direction	horizontal wind direction
r_vapor	water vapor mixing ratio
pi	total Exner function
press	pressure
theta	potential temperature
tke	prognostic turbulent kinetic energy
rain	rain mixing ratio
pristine	pristine ice mixing ratio
snow	snow mixing ratio
aggregates	aggregates mixing ratio
graupel	graupel mixing ratio
hail	hail mixing ratio
cloud	cloud water mixing ratio
liquid	cloud water plus rain mixing ratios
ice	ice mixing ratio (sum of pristine ice, snow, aggregates, graupel, and hail mixing ratios)
totpcp	total accumulated precipitation at ground
pcprate	current precipitation rate at ground
khv	vertical diffusion coefficient
khh	horizontal diffusion coefficient (not available with standard configuration)
tempc	air temperature in degrees Celsius

tempf	air temperature in degrees Fahrenheit
theta_e	equivalent potential temperature
dewptf	dew point temperature in degrees Fahrenheit
dewptc	dew point temperature in degrees Celsius
relhum	relative humidity
topo	topography height
lat	latitude
lon	longitude
pctlnd	land percentage
sea_press	sea level pressure
sst	sea surface temperature
conpcp	convective precipitation rate
acccon	accumulated convective precipitation at ground
vtype	vegetation type
sfc_div	surface horizontal divergence
tveg	vegetation temperature
tg	bare soil temperature
slmst	bare soil moisture
ts	shaded soil temperature
slmss	shaded soil moisture

Index

ZZ 61

CHAPTER 9 - CUSTOMIZING THE ATMOSPHERIC MODEL CODE: THE MOST COMMON APPLICATIONS

It is often necessary for a user to modify one or more sections of the atmospheric model code to customize it for a particular application. While the namelist variables described in Chapter 7 control the vast majority of options available in the model, it is simply not practical to place all foreseeable options under their control. We thus describe here the most commonly encountered reasons for requiring code modifications, and present the most straightforward means of accomplishing each. It is convenient to categorize the most common code modifications into groups, and each of these is discussed in a separate section of this chapter.

Section 9.1 - Customization of Initial Surface, Vegetation, and Soil Data

The ground surface bounding the bottom of the atmospheric domain possesses the following characteristics which are relevant to a general atmospheric simulation: (1) terrain height, (2) land roughness length, (3) land percentage, and (4) water surface temperature. The soil layers beneath the surface possess the following characteristics, which we number adding to the above list: (5) soil textural class, (6) soil temperature, and (7) soil moisture content. The vegetation canopy just above the ground surface possesses the following characteristics, again appended to the above list: (8) vegetation type, (9) vegetation temperature, and (10) water content (*e.g.*, rain or dew) on the vegetation surfaces. All 10 of the above characteristics must be defined for each surface grid cell in the model domain for a general simulation (if the soil/vegetation parameterization is not activated, only the first 4 characteristics are needed).

Characteristics (1) through (5) plus (8) are treated as constant in time for the duration of a simulation, and need be defined only when the simulation is initialized. By default, each of these is initialized with horizontally homogeneous values, all of which except for terrain height are specified by the user in the namelist variables ZROUGH for (2), PCTLCON for (3), SEATMP for (4), NSLCON for (5), and NVGCON for (8) (see Section 7.4). Terrain height is by default set to zero everywhere. The other characteristics are prognostic variables in the model, but require initial values at the onset of a simulation. By default, the initial values for (6) and (9) are set to horizontally-homogeneous offsets, given by STGOFF and TVGOFF, respectively, from the local atmospheric values of temperature just above the surface. Soil moisture (7) and vegetation water content (10) are by default horizontally-homogeneously set to the namelist variables SLMSTR and VWTRCON, respectively (see Section 7.4).

Each of the above 10 characteristics, except for (6) and (7), is defined in its own two-dimensional array covering all surface grid points. Characteristics (6) and (7) are properties of the soil which are height dependent, and are thus defined in three-dimensional arrays. The array names corresponding to all the characteristics are as follows: TOPT for (1), ZO for (2), PCTLND for (3), SST for (4), SLTEX for (5), TGP for (6), SLMSTP for (7), VGTYP for (8), TVGP for (9), and WRESV for (10). The actual filling of each of these arrays is performed by default in subroutines TOPTINIT, SFCINITF, and SFCINIT in the file **rsurf.f**, or by communication from values on other grids. Following this standard initialization procedure, subroutines TOPTINIT_USER, SFCINITF_USER, and SFCINIT_USER in the file **ruser.f** are called. The naming of this particular module is intended to designate the code most often requiring user intervention. These subroutines access all of the above surface, soil, and vegetation variables. It is in these subroutines that the user can change these variables before they are used in the model integration. By default, the subroutines in **ruser.f** are commented out (except for the nonexecutable parts) so they perform no function. The commented code serves as a template for customizing the assignment of individual values to the above arrays. Such customizing is commonly done by (1) defining values algebraically as a function of the horizontal grid indices (I and J) of the array or of the horizontal coordinates XT(I) and YT(J), (2) by simple assignment statements for individual points in the array, (3) by reading from an external file, or by a combination of all three. The subroutines in **ruser.f** are called once for each active grid, so the user normally wants to uncomment the FORTRAN IF-blocks to make the assignments grid-dependent.

We point out here four exceptions to the above statement that all 11 arrays are filled in the file **rsurf.f**. Terrain height TOPT, land percentage PCTLND, sea-surface temperatures SST, and vegetation type VGTYP may optionally be filled directly from specialized datasets which have been prepared for RAMS. This option is activated independently for each grid by setting the namelist flags ITOPTFLG, IPCTLFLG, ISSTFLG, and IVEGTFLG, respectively, to 1 (see Section 7.3). The alternative has been made possible by the availability of these datasets. It is anticipated that datasets for more of the 11 variables will become available in the future, and at such time, they will be made usable in RAMS in the same manner.

Section 9.2 - Customization of Initial Atmospheric Prognostic Fields

It is often desired to initialize certain atmospheric fields, such as potential temperature and moisture mixing ratio, with specific horizontally inhomogeneous perturbations. For example, a high resolution simulation of a thunderstorm may be initialized with a local region of enhanced temperature and moisture to trigger the onset of convection at a specific location. Another example is in using automatically added scalar fields (see description of namelist variable NADDSC in Section 7.4), where initial values of the scalar field are required to be nonzero. Sometimes it is desirable to add a perturbation to a prognostic field at a later time, after a simulation has run for a period of time.

The following template subroutine, called subroutine PERT, is provided in the file ruser.f in order to customize one or more prognostic fields in the model.

```
      subroutine pert(n1,n2,n3,thp,rtp)
      dimension thp(n1,n2,n3),rtp(n1,n2,n3)
c
c     do j = 1,1
c        do i = 17,26
c           do k = 2,7
c              thp(k,i,j) = thp(k,i,j) + 5.
c              rtp(k,i,j) = rtp(k,i,j) * 1.2
c           enddo
c        enddo
c     enddo
      return
      end
```

In this example, two 3-D arrays, potential temperature (thp) and total water mixing ratio (rtp) are passed to the subroutine, as are the dimensions (n1,n2,n3) of the arrays. The subroutine's executable lines are commented out by default, but illustrate an example where thp is increased by 5 K and rtp is increased by 20% in the lowest 6 model levels (k=2,7) and over 10 different locations in the x-direction (i=17,26).

In order to allow the flexibility to place a perturbation on any or all grids and at any desired time, it is most convenient to insert a call to subroutine PERT in subroutine TIMESTEP in the **rtimh.f** file. A good place is to add the call is before the call to TEND0. Note that there are three sections to subroutine TIMESTEP, and only one is executed depending on the value of ITMDIFF set in the namelist (see Chapter 7). Thus, add a call such as the following in the appropriate section:

```
      if (time .le. 0.1 .and. ngrid .eq. 2)
     +   call pert(nzp,nxp,nyp,a(ithp),a(irtp))
```

In this example, the call is executed only at the initial model time and for grid number 2. Note that the call to subroutine PERT passes address ITHP in the A array. The A array contains all prognostic variables in the model, as well as many other arrays. To place an initial perturbation in the velocity component fields, the appropriate A array indices to place in the subroutine call are IUP, IVP, and/or IWP. For automatically added scalar fields, the A array indices are ISCLP(1), ISCLP(2), etc. Specification of the perturbation is often done as an algebraic function of the grid indices (K,I,J) or coordinates ZT(K), XT(I), and YT(J) (which are available in the 'rcommons.h' common block).

Section 9.3 - Adding a New Source Term to Model Prognostic Fields

Another need which sometimes arises in the use of the atmospheric model is to add an artificial forcing term to one or more prognostic fields. This is common for the potential temperature and moisture fields, and particularly for added scalar fields (see description of namelist variable NADDSC in Section 7.4). For example, as an alternative to placing a warm, moist perturbation in the initial fields to trigger convection as described in the previous section, a heat and moisture *source* may instead be imposed in a region of the domain. This source adds the heat and moisture gradually over a period of time.

The following template subroutine, called subroutine FORCE, is provided in the file ruser.f in order to customize one or more prognostic fields in the model.

```
      subroutine force(n1,n2,n3,tht)
      dimension tht(n1,n2,n3)
c
c     do j = 1,1
c        do i = 17,26
c           do k = 2,7
c              tht(k,i,j) = tht(k,i,j) + 0.001
c           enddo
c        enddo
c     enddo
      return
      end
```

In this example which is very similar to the one in Section 9.2, the 3-D array **tht**, which is the potential temperature tendency in [K/s], is given a value of 0.001.

As for the application described in Section 9.2, this is most conveniently done from subroutine TIMESTEP by adding a call such as the following *after* the call to TEND0 (note as above the three sections in subroutine TIMESTEP):

```
         if(time .lt. 1800.) call force(nzp,nxp,nyp,a(itht))
```

In this example, the forcing term is applied to **tht** only during the first 1800 seconds of the simulation. The "A" array contains all tendency variables in the model, as well as many other arrays. To place a perturbation in the velocity component tendencies, the appropriate "A" array indices to place in the subroutine call are IUT, IVT, and/or IWT, and for added scalars, the "A" array indices for tendency are ISCLT(1), ISCLT(2), etc.

Section 9.4 - Adding Array Space to the Model

In some applications, a user needs to dimension a new array in the atmospheric model. This Section describes four different ways this might be done. Depending on the specific purpose of the new array, any one of these ways may be better than the others.

1. If the new array space is only required within one subroutine, it is perhaps best to simply add the appropriate "DIMENSION" statement to that subroutine. We point out, however, that if the new array is to be used only as scratch space whose retention is not required following return from the subroutine, the user should consider using the existing alternate scratch space in the model to save on memory. This scratch space consists of blocks in the "A" array, each having at least as much space as the number of cells in the active model grid. For example, in the call to the subroutine, add the argument A(ISCR1), and in the subroutine itself dimension the corresponding new argument up to the size of the grid. The other indices locating scratch space within the "A" array are ISCR2, IVT3DA, IVT3DB, IVT3DC, IVT3DD, IVT3DE, IVT3DF, IVT3DG, IVT3DH, IVT3DI, IVT3DJ, and IVT3DK. An example follows. In the calling subroutine you would have

   ```
   CALL MYCALC(NZP, NXP, NYP, A(ISCR1))
   ```

   Then the subroutine would look like

   ```
   SUBROUTINE MYCALC(N1, N2, N3, VAR)

   DIMENSION VAR(N1,N2,N3)
     .
     .
     .
   ```

2. If the required new array space must be retained, say from one model timestep to the next (*i.e.*, it cannot be used as scratch space), and if in addition the new array is needed in two or more different subroutines, it may be best to define a new labeled common block and insert it in all the subroutines requiring the array space.

3. If the new array to be added represents a scalar field in the model, for example a mixing ratio of a trace gas which needs to be advected and diffused along with other model scalars, then by far the best method of adding the array is to incorporate it within the "A" array; doing so in the appropriate manner makes its treatment as a prognostic scalar field automatic. This version of RAMS now has an automatic way of adding scalar fields (see NADDSC in section 7.4).

4. If the new array is *not* a prognostic field and must be retained from one model timestep to the next (*i.e.*, it cannot be used as scratch space), and particularly if it is desired to save the new array on history and/or analysis files, then it needs to be entered manually into the A array. Version 3b of RAMS happens to have up to ten available locations in the memory tables allowing this to be done very easily. If more than ten additional variables are required, the procedure is more complicated. Discussion here will be limited to cases where the number of new variables in this class does not exceed ten. Let's actually follow the example where all ten arrays will be added. All that is required in this case to add the space to the A array is to make the following changes to SUBROUTINE VTABLES (in **rvtab.f**).

   Find the following lines:

   ```
   IVTAB3D(NV3D(NG)+9,NG)=21
   NV3D(NG)=NV3D(NG)+9
   ```

   Replace the second of these lines with the lines:

```
IVTAB3D(NV3D(NG)+10,NG)=51
IVTAB3D(NV3D(NG)+11,NG)=52
IVTAB3D(NV3D(NG)+12,NG)=53
IVTAB3D(NV3D(NG)+13,NG)=54
IVTAB3D(NV3D(NG)+14,NG)=55
IVTAB3D(NV3D(NG)+15,NG)=56
IVTAB3D(NV3D(NG)+16,NG)=57
IVTAB3D(NV3D(NG)+17,NG)=58
IVTAB3D(NV3D(NG)+18,NG)=59
IVTAB3D(NV3D(NG)+19,NG)=60
NV3D(NG)=NV3D(NG)+19
```

In the above group of lines, each line except for the last represents one of the new fields added (a total of 10 in this example). The numbers on the right hand sides of the equals signs are the variable numbers (order of occurrence) in the memory tables of the available ten new variables. These particular memory table numbers were formerly used for adding prognostic scalar variables, now handled automatically by the NADDSC namelist variable (see above and in Section 7.4).

The last line in the above group is now incremented by 19, rather than 9 as before, to reflect the 10 new fields added. This accomplishes adding the variables to memory, and they are accessed in the model as A(IDUM1), A(IDUM2), ... A(IDUM10).

If it is required to save the new arrays on history files, then find the following lines:

```
IHTAB3D(NH3D(NG)+5,NG)=7
NH3D(NG)=NH3D(NG)+5
```

and replace the second of these lines with the lines:

```
IHTAB3D(NH3D(NG)+6,NG)=51
IHTAB3D(NH3D(NG)+7,NG)=52
IHTAB3D(NH3D(NG)+8,NG)=53
IHTAB3D(NH3D(NG)+9,NG)=54
IHTAB3D(NH3D(NG)+10,NG)=55
IHTAB3D(NH3D(NG)+11,NG)=56
IHTAB3D(NH3D(NG)+12,NG)=57
IHTAB3D(NH3D(NG)+13,NG)=58
IHTAB3D(NH3D(NG)+14,NG)=59
IHTAB3D(NH3D(NG)+15,NG)=60
NH3D(NG)=NH3D(NG)+15
```

Similarly, in order to write the new arrays to analysis files, then find the following lines:

```
IATAB3D(NA3D(NG)+5,NG)=19
NA3D(NG)=NA3D(NG)+5
```

and replace the second of these lines with the lines:

```
IATAB3D(NA3D(NG)+6,NG)=51
IATAB3D(NA3D(NG)+7,NG)=52
IATAB3D(NA3D(NG)+8,NG)=53
IATAB3D(NA3D(NG)+9,NG)=54
IATAB3D(NA3D(NG)+10,NG)=55
IATAB3D(NA3D(NG)+11,NG)=56
IATAB3D(NA3D(NG)+12,NG)=57
IATAB3D(NA3D(NG)+13,NG)=58
IATAB3D(NA3D(NG)+14,NG)=59
IATAB3D(NA3D(NG)+15,NG)=60
NA3D(NG)=NA3D(NG)+15
```

The above two steps added the the ten new arrays to the history and analysis tables, so that they would be written to the history and analysis files output by the model.

Section 9.5 - Adding a Variable to be Plotted in VAN or REVU

The easiest way to add a variable to be plotted in VAN is to use the existing structure in FUNCTION ANALIB in **rafld.f** as a template. This function is essentially a series of ELSEIF blocks, with each variable to be plotted contained within one of the blocks. There are two main types of variables that you may want to plot, those quantities that are saved as part of the analysis files and those that must be calculated from quantities saved in the analysis files. We will consider examples of each, but if you are not familiar with the structure of FUNCTION ANALIB, take a few minutes to look at the first several lines of code, for say, U, V, and W. Note that in each block of code, ANALIB is given the value returned by FUNCTION VALUGP. This important function simply returns the value of the array (specified by the last argument in the function call) at the grid location specified by K,II,JJ (see example below). The first three arguments of the function call are the three spatial dimensions of the array space.

As a first example, suppose you would like plots of total Exner function. Since this is saved in the analysis files, the following code would be added to FUNCTION ANALIB:

```
      ELSEIF (FVAR.EQ.'PITOT') THEN
         IF(K.LT.0)THEN
            TITLE ='total Exner function'
            CPOINT='T'
         ELSEIF(K.EQ.0)THEN

         ELSE
            ANALIB=VALUGP(NNZP(NGRID),NNXP(NGRID),NNYP(NGRID),K,II,
     .           JJ,A(IPP))
         ENDIF
```

The first line of code specifies the variable name, 'PITOT', which you would then specify in VAN namelist, e.g., CCONTOUR(1,2)='PITOT', if 'PITOT' is the second plot on the first background (see Section 8.2). The 'TITLE' is specified by the user.

The second example is also relatively simple. Suppose you would like to see THETA in degrees C instead of degrees K. The code would be as follows:

```
      ELSEIF (FVAR.EQ.'THETAC') THEN
         IF(K.LT.0)THEN
```

```
            TITLE ='potential temp C'
            CPOINT='T'
         ELSEIF(K.EQ.0)THEN

         ELSE
            ANALIB=VALUGP(NNZP(NGRID),NNXP(NGRID),NNYP(NGRID),K,II,
     .            JJ,A(ITHETA)) - 273.15
         ENDIF
```

The differences from plotting THETA in degrees K are: (1) different FVAR, (2) different TITLE, and (3) ANALIB is now modified by 273.15 since K - 273.15 = C.

The final example is to calculate divergence ($\partial u/\partial x + \partial v/\partial y$) and is more complicated in that it involves calling another subroutine to calculate the desired quantity. The first block of code follows:

```
      ELSEIF (FVAR.EQ.'DIVERG') THEN
         IF(K.LT.0)THEN
            TITLE='horizontal divergence'
            CPOINT='T'
         ELSEIF(K.EQ.0)THEN
            CALL DIVCALC(NZP,NXP,NYP,A(1),A(IUP),A(IVP),
     +                   A(IVT3DA),A(IVT3DB),A)
         ELSE
            ANALIB = VALUGP(NZP,NXP,NYP,K,II,JJ,A(1))
         ENDIF
```

Note that there is a call to DIVCALC, a subroutine that calls other library routines to calculate the desired gradients, and then combine these. The address A(1) is passed out of SUBROUTINE DIVCALC and this is then plotted in the variable ANALIB. This subroutine could be attached to the end of **rafld.f** and would take the following form:

```
      SUBROUTINE DIVCALC(N1,N2,N3,TRM,UP,VP,VT3DA,VT3DB,A)
      DIMENSION TRM(N1,N2,N3),UP(N1,N2,N3),VP(N1,N2,N3)
     +         ,VT3DA(N1,N2,N3),VT3DB(N1,N2,N3),A(*)
      include 'rcommons.h'
C
      CALL GRAD(NZP,NXP,NYP,NZP,NXP,NYP,N2X,NX,N2Y,NY,0,0
     +   ,UP,VT3DA,'XDIR','UPNT',A)
      CALL GRAD(NZP,NXP,NYP,NZP,NXP,NYP,N2X,NX,N2Y,NY,0,0
     +   ,VP,VT3DB,'YDIR','VPNT',A)
      CALL AE1P1(NXYZP,TRM,VT3DA,VT3DB)
C
      RETURN
      END
```

Numerous other examples can be found by studying **rafld.f**. Note that although our example with divergence called a subroutine that used RAMS library routines, this will likely not always be the case, and the subroutine for your needs may include user-defined calculations.

In order to add a variable to be plotted in REVU, a similar procedure to that described above would be carried out. REVU's counterpart to VAN's file **rafld.f** is the file **hvlib.f** in the **mrc/post** directory.

Section 9.6 - Adding a Variable to be Printed from the $MODEL_PRINT Namelist

The procedure for adding a field to be printed out follows the same logic as we followed in Section 9.5, except that we're dealing with different routines.

Again we will start with a simple example. Suppose you would like to print out temperatures in degrees F. In **rprnt.f**, there is a FUNCTION OPTLIB, which is similar to FUNCTION ANALIB. There are ELSEIF blocks for each variable that you might want to print. For temperatures in degrees F, you would want to add a section of code as follows:

```
C   Temperature in degrees F

      ELSEIF(VARN.EQ.'TEMPF') THEN
        IF(K.EQ.0)THEN
          IPLGRD=4
          FMT='0PF7.2'
          TIL='TEMP (F)'
        ELSE
          VALTHET = VALUGP(NZP,NXP,NYP,K,I,J,A(ITHETA))
          VALPP   = VALUGP(NZP,NXP,NYP,K,I,J,A(IPP))
          VALPIO  = VALUGP(NZP,NXP,NYP,K,I,J,A(IPIO))
          OPTLIB  = (9./5.)*(VALTHET*(VALPP+VALPIO)/CP -273.15) + 32.
        ENDIF
C
```

Three changes were made from the 'ELSEIF' block for temperature in degrees K. First VARN is set to a user-defined string. Note that you will need to specify this exact string (i.e., IPLFLD='TEMPF',) in the **$MODEL_PRINT** namelist. Second, TIL is now also user specified. Third, OPTLIB has been modified to change from degrees K to degrees F.

Now, as a somewhat more complicated example, suppose we want to print out values of divergence. The block of code needed would be as follows:

```
C   divergence

      ELSEIF(VARN.EQ.'DIV') THEN
        IF(K.EQ.0)THEN
          IPLGRD=4
          FMT='0PF7.2'
          TIL='DIVERGENCE'
          CALL DIVCALC(NZP,NXP,NYP,A(1),A(IUP),A(IVP),
     +                  A(IVT3DA),A(IVT3DB),A)
        ELSE
          OPTLIB  = VALUGP(NZP,NXP,NYP,K,I,J,A(1))
        ENDIF
C
```

Subroutine DIVCALC (the same one as used in Section 9.5) will need to added to the end of **rprnt.f**, and as before, DIV needs to specified in the **$MODEL_PRINT** namelist.