EPA/600/8-90/083c
JANUARY 1991

THE REGIONAL OXIDANT MODEL (ROM) USER'S GUIDE
PART 3:
THE CORE MODEL

by

J. YOUNG
L. MILICH
D. JORGE

Computer Sciences Corporation
Research Triangle Park, NC 27709

Contract No. 68-01-7365

Technical Monitor

Joan Novak*
Atmospheric Sciences Modeling Division
Atmospheric Research and Exposure Assessment Laboratory
Research Triangle Park, NC 27711

*On assignment from the National Oceanic and Atmospheric Administration, U.S. Department of Commerce

ATMOSPHERIC RESEARCH AND EXPOSURE ASSESSMENT LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
RESEARCH TRIANGLE PARK, NC 27711

EPA/600/8-90/083c
JANUARY 1991

THE REGIONAL OXIDANT MODEL (ROM) USER'S GUIDE PART 3: THE CORE MODEL

by

J. YOUNG
L. MILICH
D. JORGE

Computer Sciences Corporation
Research Triangle Park, NC 27709

Contract No. 68-01-7365

Technical Monitor

Joan Novak*
Atmospheric Sciences Modeling Division
Atmospheric Research and Exposure Assessment Laboratory
Research Triangle Park, NC 27711

*On assignment from the National Oceanic and Atmospheric Administration, U.S. Department of Commerce

ATMOSPHERIC RESEARCH AND EXPOSURE ASSESSMENT LABORATORY
OFFICE OF RESEARCH AND DEVELOPMENT
U.S. ENVIRONMENTAL PROTECTION AGENCY
RESEARCH TRIANGLE PARK, NC 27711

Mention of trade names or commercial products
does not constitute endorsement or recommendation for use.

TABLE OF CONTENTS

Disclaimer ii
Table of Contents iii
List of Tables v
List of Figures viii

1 Overview and Structure of the Core Model 1
1.1 Introduction 1
1.2 General Model Characteristics 3
1.2.1 Physical Processes Within Layers 0, 1, 2 and 3 4
1.2.2 ROM Chemistry 7
1.2.3 System Components 8
1.2.4 ROM Limitations 11
1.3 Data and File Requirements 11
1.4 Model Execution and Storage Requirements for the IBM 3090 13
1.4.1 Starting the ROM at the Top of an Hour 13
1.4.2 CPU Use 13
1.4.3 Storage Requirements 14
1.4.4 Core Model Input Data: Transfer from the VAX to the IBM 14
1.5 Software Components of the Core Model 15
1.5.1 BIGGAM 16
1.5.2 LILGAM 19
1.6 Subprograms That Maintain the State Vector and Restart the Model 22
1.7 Utility Subprograms 22
1.8 Miscellaneous Subprograms 24
1.9 Core Model Output Data: Transfer from the IBM to the VAX 25
1.10 CONC File Quality Control Procedures 25

2 Core Model Input Files 27
2.1 The Boundary Conditions (BCON) File 27
2.1.1 Opening the BCON File 28
2.1.2 BCON File Records 29
2.2 The B-Matrix (BMAT) File 34
2.2.1 Opening the BMAT File 35
2.2.2 BMAT File Records 36
2.3 The Backtrack (BTRK) File 48
2.3.1 Opening the BTRK File 48
2.3.2 BTRK File Records 49
2.4 The Initial Conditions (ICON) File 54
2.4.1 Opening the ICON File 55
2.4.2 ICON File Records 56

2.5 The New Initial Conditions (NEWICON) File 60
2.5.1 Opening the NEWICON File 61
2.5.2 NEWICON File Records 62
2.6 The Progress (PROG) File 68
2.6.1 Opening the PROG File 68
2.6.2 The PROG File Record 69
2.7 The State Vector (RESTRT) File 69
2.7.1 Processing That Takes Place for Normal Model Execution (How the RESTRT File Gets Written) 70
2.7.2 Processing That Takes Place for Restarting Model Execution (How the RESTRT File Gets Read) 85
2.8 The Stop Check (STOPCK) File 88
2.8.1 Opening the STOPCK File 88
2.8.2 The STOPCK File Record 89

3 The Core Model Concentration (CONC) Output File 91
3.1 Opening the CONC File 91
3.2 CONC File Records 92
3.2.1 CONC File Header Records 93
3.2.2 CONC File Body Records 97

APPENDICES

A Jackson Structured Programming (JSP) A-1
A.1 An Introduction to JSP Flow Diagrams A-2
A.2 Examples of Program Inversion and the Use of State Vectors A-4
A.2.1 Program Inversion with Respect to Its Input Data Stream A-5
A.2.2 Program Inversion with Respect to Its Output Data Stream A-6
A.3 Reference and Bibliography A-6

B Design and Structure Diagrams for the ROM2.1 Data Files in the ROMNET Region B-1

C Design and Structure Diagrams for the Principal ROM2.1 Subroutines C-1

D Sample Compile and Link Stream for the Uniprocessor ROM2.1 ..... D-1

E INCLUDE Files E-1

F Core Model Error Checking F-1

Index Index-1

LIST OF TABLES

1 ROM chemical species 8
2 HEADIN.EXT variables: descriptions and origins 18
3 BCON record 1 variables 30
4 BCON record 2 variable 31
5 BCON record 3 variable 31
6 BCON records 4 - (4+ICNTBM) variable 32
7 BCON time-step header record variables 32
8 BCON data variables 34
9 B-matrix record 1 variables 38
10 B-matrix record 2 variable 38
11 B-matrix record 3 variable 40
12 B-matrix record 4 variable 40
13 B-matrix records 5 - 8 variables 41
14 B-matrix records 9 - (9+ICNTBM-1) variables 41
15 B-matrix subfile-order record variables 42
16 B-matrix time-step header record variables 43
17 B-matrix data variables, part 1 46
18 B-matrix data variables, Part 2 48
19 BTRK record 1 variables 51
20 BTRK records 2 - 5 variables 52
21 BTRK records 6 - (6+ICNTBT-1) variables 52
22 BTRK time-step header record variables 53
23 BTRK data variables 54
24 ICON record 1 variables 57
25 ICON record 2 variable 58
26 ICON record 3 variable 58
27 ICON records 4 - (4+ICNTIC) variable 59
28 ICON time-step header record variables 59
29 ICON data variable 60
30 NEWICON record 1 variables 64
31 NEWICON record 2 variable 65
32 NEWICON record 3 variable 65
33 NEWICON records 4 - (4+ICNTCN) variable 66
34 NEWICON time-step header record variables 67
35 NEWICON data variable 68
36 PROG record variables 69
37 RESTRT record 1 variables 73
38 RESTRT record 2 variables 74
39 RESTRT records 3 and 4 variable 74
40 RESTRT record 5 variable 75

41 RESTRT records 6 - (6+ICNTSV) variable 76
42 RESTRT HEADIN record 1 variables 76
43 RESTRT HEADIN record 2 variables 77
44 RESTRT chemistry control variables 1 78
45 RESTRT chemistry control variables 2 78
46 BMAT species index variable 79
47 BCON species index variable 79
48 ICON species index variable 80
49 Primary oxidant species flag records variable 80
50 Special species record variables 81
51 Diffusivities conversion factor record variables 82
52 ICON time-step header record variables 82
53 Text pointer record variables 83
54 Row counters record variables 84
55 Model and file time step header records 85
56 STOPCK record variable 89
57 CONC record 1 variables 95
58 CONC record 2 variable 96
59 CONC record 3 variable 97
60 CONC records 4 - (4+ICNTCN) variable 97
61 CONC time-step header record variables 98
62 CONC data variable 99
E-1 INCLUDE file DIMENS.EXT E-2
E-2 INCLUDE file REGION.EXT E-2
E-3 INCLUDE file ADVSFL.EXT E-3
E-4 INCLUDE file BCFILE.EXT E-3
E-5 INCLUDE file BGBCFL.EXT E-3
E-6 INCLUDE file BGBTFL.EXT E-4
E-7 INCLUDE file BGICCN.EXT E-4
E-8 INCLUDE file BMCOEF.EXT E-5
E-9 INCLUDE file BMFILE.EXT E-6
E-10 INCLUDE file BTFILE.EXT E-7
E-11 INCLUDE file CHEMIN.EXT E-7
E-12 INCLUDE file CHEMSW.EXT E-8
E-13 INCLUDE file CNFILE.EXT E-8
E-14 INCLUDE file CONFAC.EXT E-8
E-15 INCLUDE file ERRG.EXT E-9
E-16 INCLUDE file FLNAMS.EXT E-9
E-17 INCLUDE file GTCOEF.EXT E-10
E-18 INCLUDE file HDFMTS.EXT E-10
E-19 INCLUDE file HDSTAV.EXT E-11
E-20 INCLUDE file HEADBC.EXT E-12

E-21 INCLUDE file HEADBM.EXT E-13
E-22 INCLUDE file HEADBT.EXT E-14
E-23 INCLUDE file HEADCN.EXT E-15
E-24 INCLUDE file HEADIC.EXT E-16
E-25 INCLUDE file HEADIN.EXT E-17
E-26 INCLUDE file HSTEPS.EXT E-18
E-27 INCLUDE file ICFILE.EXT E-18
E-28 INCLUDE file LGBMFL.EXT E-19
E-29 INCLUDE file LILGSP.EXT E-20
E-30 INCLUDE file LUNITS.EXT E-20
E-31 INCLUDE file LVNAME.EXT E-21
E-32 INCLUDE file NDXPC.EXT E-21
E-33 INCLUDE file NROOTS.EXT E-21
E-34 INCLUDE file RKLEVS.EXT E-22
E-35 INCLUDE file ROWSCT.EXT E-22
E-36 INCLUDE file RTCONS.EXT E-23
E-37 INCLUDE file RTSHBC.EXT E-23
E-38 INCLUDE file RTSHBM.EXT E-24
E-39 INCLUDE file RTSHBT.EXT E-24
E-40 INCLUDE file RTSHCN.EXT E-25
E-41 INCLUDE file RTSHIC.EXT E-25
E-42 INCLUDE file RUNTMS.EXT E-26
E-43 INCLUDE file SPNAME.EXT E-26
E-44 INCLUDE file STOPFL.EXT E-26
E-45 INCLUDE file SUBID.EXT E-27
E-46 INCLUDE file TEXTPT.EXT E-28
E-47 INCLUDE file TILDE.EXT E-28
E-48 INCLUDE file TSHDBC.EXT E-29
E-49 INCLUDE file TSHDBM.EXT E-29
E-50 INCLUDE file TSHDBT.EXT E-29
E-51 INCLUDE file TSHDCN.EXT E-30
E-52 INCLUDE file TSHDIC.EXT E-30
E-53 INCLUDE file TSHDMD.EXT E-30
E-54 INCLUDE file TSHDSV.EXT E-31
E-55 INCLUDE file TSTEPS.EXT E-31
E-56 INCLUDE file UNITIO.EXT E-32
E-57 INCLUDE file file ZADVSL.EXT E-32

LIST OF FIGURES

1 NEROS, SEROS, and ROMNET modeling domains 4
2 The ROM vertical layers and their functional features 5
3 The principal components of the ROM 10
4 Conceptual data-flow diagram of the Core Model 15
5 BIGGAM subprograms 17
6 LILGAM subprograms 20
7 Chemical species mapping from BMAT to the Core Model 39
B-1 The BCON file B-2
B-2 The BMAT file B-5
B-3 The BTRK file B-10
B-4 The ICON file B-13
B-5 The CONC file B-16
C-1 Core Model system specification diagram C-2
C-2 Core Model Jackson Structured Design implementation diagram C-3
C-3 RUNMGR C-4
C-4 BIGGAM C-6
C-5 ICPRCS C-7
C-6 RDICON C-8
C-7 BCPRCS C-9
C-8 RDBCON C-10
C-9 BTPRCS C-11
C-10 RDBTRK C-12
C-11 RDBT C-13
C-12 LILGAM C-14
C-13 BMPRCS C-16
C-14 RDBMAT C-17
C-15 RDMXBM C-18
C-16 CNPRCS C-19
C-17 WRCONC C-20
C-18 RDCONC C-21

SECTION 1

OVERVIEW AND STRUCTURE OF THE CORE MODEL

1.1 INTRODUCTION

The initial development of a regional (≈1000km) air quality simulation model began in the late 1970's after the realization that photochemical smog often extended beyond individual urban areas to entire sections of the United States. Interstate transport of ozone (O_3) and its precursors was observed during field programs, especially in the northeast of the country. Since multiday chemical effects and long-range transport of ozone and its precursors was beyond the scope of the existing urban-scale photochemical models, the need for an appropriate simulation model to test the effectiveness of particular emission control strategies on regional and urban airshed ozone concentrations became clear. The Regional Oxidant Model (ROM) has been developed and enhanced over the past eight years by the U.S. Environmental Protection Agency (EPA) in response to this need. The first generation ROM (ROM1.0) became operational in 1984. The initial model formulation and algorithm testing is documented in a three-part volume titled *A Regional Scale (1000 km) Model of Photochemical Air Pollution*: Part 1 - *Theoretical Formulation*; Part 2 - *Input Processor Network Design*; Part 3 - *Tests of Numerical Algorithms*.[1]

ROM1.0 was a test case for future production versions of the model. It contained a very condensed chemical kinetics mechanism, did not treat natural hydrocarbon emissions, terrain effects, or the vertical mass flux induced by clouds, and used constant rather than dynamic layer depths. ROM2.0 became operational in 1987, and included a more sophisticated, contemporary chemical kinetics mechanism capable of treating both anthropogenic and biogenic precursor species. ROM2.0 also corrected many of the deficiencies and simplifications of ROM1.0, such as using variable layer thicknesses and properly treating cloud-induced mass flux and terrain effects. The current version of the ROM, ROM2.1, became operational in 1989, and was developed chiefly in response to the needs of the EPA's Regional Ozone Modeling for Northeast Transport (ROMNET; EPA, 1990) project. ROM2.1 allows relatively simple changes in the code to increase or decrease the modeling domain's size, and is adaptable to other modeling domains in eastern North America. Some other modifications include use of an upgraded chemical kinetics mechanism, an updated biogenic hydrocarbon processor, and expanded use of meteorology and

1. Lamb, 1983, Lamb, 1984a, and Lamb and Laniak, 1985, respectively.

anthropogenic emissions data. All new features and modifications of the ROM2.1 are documented in Young *et al.* (1990). The EPA is continuing to upgrade the ROM; version 2.2 will likely become operational in 1991.

The ROM was primarily developed in a VAX environment. Although software development proceeded in tandem with hardware development - with migrations of the code to more powerful machines as they were installed at the EPA - by the time of the ROM2.0 era, roughly 100 VAX CPU hours were required for each 3-day simulation. The model was therefore migrated to the much more powerful IBM 3090, resulting in the data transfer issues that are discussed in Sections 1.4.4 and 1.9. Even on the IBM 3090, the ROM2.0 required 6 to 8 CPU hours for a 3-day simulation, increasing to around 10 CPU hours for the ROM2.1. The CPU requirements implied that we could not be assured of one ROM2.1 execution per night, thus prompting attempts at parallelizing and vectorizing the model. Vectorization was not considered to be an appropriate strategy for reducing elapsed clock time since it decreased run time by only 20 percent. This small reduction is due mainly to (1) the "short" vectors that result from the chemistry solver code, and (2) the prodigious amount of indirect addressing in the code. However, parallelization on the six available CPUs in the IBM 3090 resulted in nearly a four-fold reduction in elapsed clock time for a model execution. The model we run at the EPA is this parallelized variant of ROM2.1; note that we are documenting the uniprocessor variant in this User's Guide. We include a sample compile and link stream for the uniprocessor variant in Appendix D.

This volume of the ROM2.1 User's Guide is intended for the programmers who will install and execute the ROM, and provides the information needed to understand the operation of the ROM2.1 Core Model.[2] Companion volumes of the ROM2.1 User's Guide describe the preprocessing of the raw meteorology and emissions input data (Part 1), and the operation of the ROM Processor Network (Part 2). The Processor Network produces the four data files required for Core Model execution: BMAT, BCON, ICON, and BTRK. Section 1.3 summarizes these files, plus the other four input files and one output file that must pre-exist, and that the Core Model expects to find. In addition, you will find the computer storage requirements listed in Section 1.4. We show how we transfer data between the VAX and IBM, and vice versa, in Sections 1.4.4 and 1.9 respectively. We describe the software structure of the model in Sections 1.5 to 1.8. Section 2 is an in-depth look at the nine input files, and Section 3 describes the final product of the ROM - the chemical concentration predictions contained in the CONC file.

Appendix A is a brief tutorial to Jackson Structured Programming and *state vectors*. We use these concepts in the Core Model code, which consists of 66 subprograms in addition to the main program, RUNMGR. Of these, 14 are variable-state subroutines, which we call *processes* to distinguish them from

2. This volume will also be useful to those people who may wish to maintain and enhance the code.

procedures. Procedures execute their code from top to bottom whenever they are invoked. Processes interact with their calling programs in a critical time sequence, hence their implementation as variable-state subroutines. The processes maintain variable text pointers in their code that are set when the process is suspended and control is returned to the calling program. At the next invocation of the process, it resumes execution of the code from the point at which it was previously suspended.

Appendix B contains design and structure diagrams for the ROM2.1 data files in the ROMNET region. Appendix C contains design and structure diagrams for the principal ROM2.1 subroutines. Appendix E contains descriptions of all the common blocks in all the INCLUDE files, including DIMENS.EXT and REGION.EXT, which are the common blocks that set the domain- and model-specific variables, and are shown in Tables E-1 and E-2 respectively. Appendix F is an in-depth guide to the Core Model's error-checking procedures.

1.2 GENERAL MODEL CHARACTERISTICS

The ROM was designed to simulate most of the important chemical and physical processes that are responsible for the photochemical production of ozone over a domain of 1000 km and for multiple 3-day episodes up to approximately 15 days in duration. These processes include (1) horizontal transport, (2) atmospheric chemistry and subgrid-scale chemical processes, (3) nighttime wind shear and turbulence associated with the low-level nocturnal jet, (4) the effects of cumulus clouds on vertical mass transport and photochemical reaction rates, (5) mesoscale vertical motions induced by terrain and the large-scale flow, (6) terrain effects on advection, diffusion, and deposition, (7) emissions of natural and anthropogenic ozone precursors, and (8) dry deposition. The processes are mathematically simulated in a three-dimensional Eulerian model with $3^1/_2$ vertical layers, including the boundary layer and the capping inversion or cloud layer.[3] Horizontal grid resolution is $^1/_4°$ longitude by $^1/_6°$ of latitude, or about 18.5 km x 18.5 km. Current model domains include the northeastern United States and the southeastern U.S./Gulf Coast area (Figure 1). For each of these domains, the model uses Eastern Standard Time (EST) in all its calculations.

3. Layer 0, the "half" layer, has a layer thickness that is always $^1/_{10}$ of the thickness of layer 1; the thicknesses of layers 1, 2, and 3 vary.

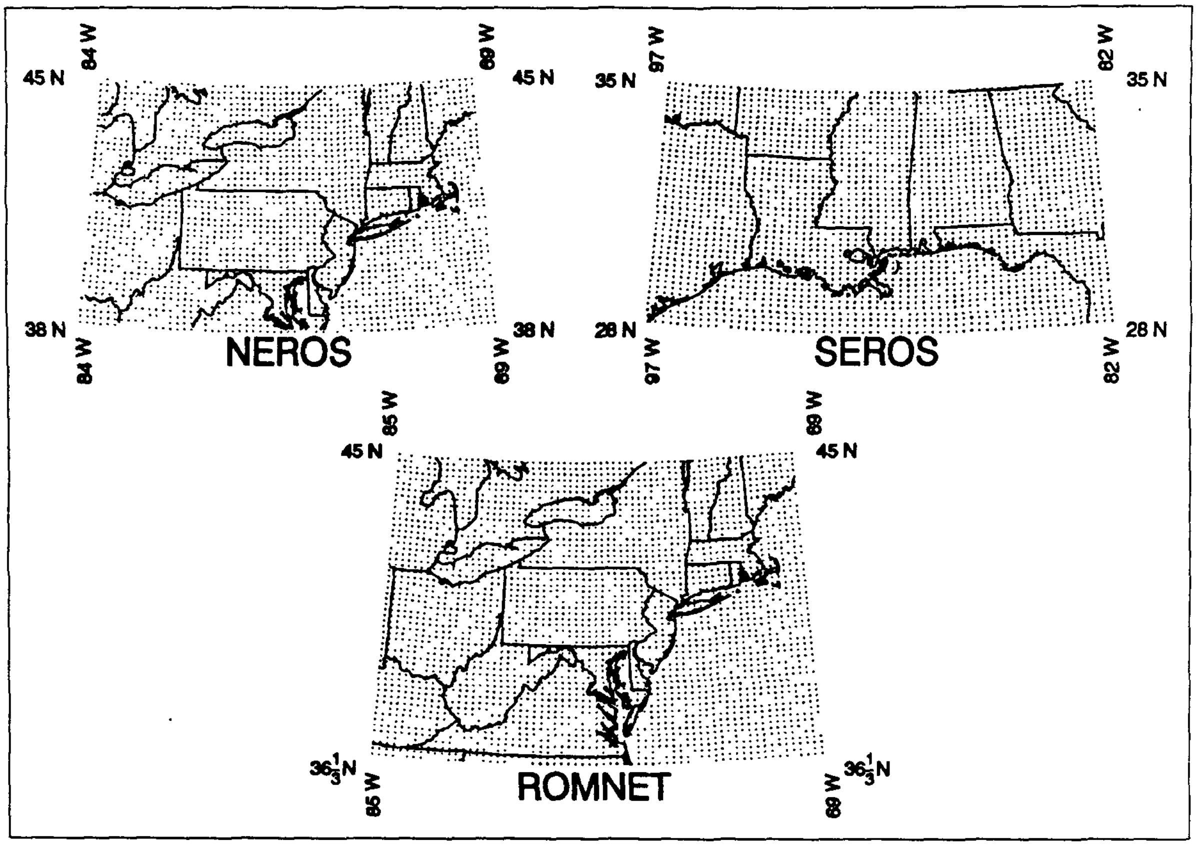


Figure 1. NEROS, SEROS, and ROMNET modeling domains for the ROM; dots represent grid cell corners.

1.2.1 Physical Processes within Layers 0, 1, 2 and 3

The meteorological data are used to objectively model regional winds and diffusion. The top three model layers are prognostic (predictive) and are free to locally expand and contract in response to changes in the physical processes occurring within them. During an entire simulation period, horizontal advection and diffusion and gas-phase chemistry are modeled in the upper three layers. The bottom layer, layer 0, is a shallow diagnostic surface layer designed to approximate the subgrid-scale effects on chemical reaction rates from a spatially heterogeneous emissions distribution. ROM predictions from layer 1 are used as surrogates for surface concentrations. The time scale of output concentrations is 30 minutes, although typically 1- and 8-hour daytime averages are used for the analysis of air quality by the public policy sector. Figure 2 shows the ROM layers during the day and at night, and describes some of their features.

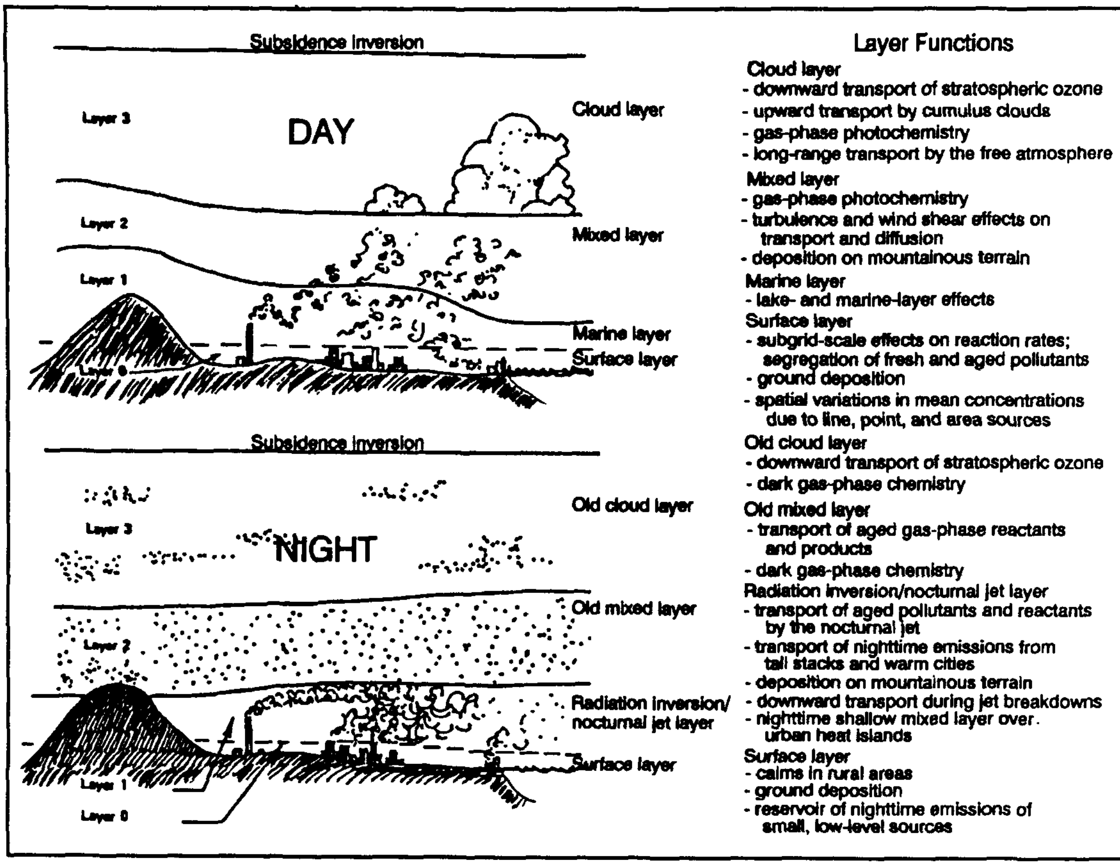


Figure 2. The ROM vertical layers and their functional features.

Layers 1 and 2 model the depth of the well-mixed layer during the day. Some special features of layer 1 include the modeling of (1) the substantial wind shear that can exist in the lowest few hundred meters above ground in local areas where strong winds exist and the surface heat flux is weak, (2) the thermal internal boundary layer that often exists over large lakes or near sea coasts, and (3) deposition onto terrain features that protrude above the layer. At night, layer 2 represents what remains of the daytime mixed layer. As stable layers form near the ground and suppress turbulent vertical mixing, a nocturnal jet forms above the stable layer and can transport aged pollutant products and reactants considerable distances. At night, emissions from tall stacks and warm cities are injected directly into layers 1 and 2. Surface emissions are specified as a mass flux through the bottom of layer 1.

During the day, the top model layer, layer 3, represents the synoptic-scale subsidence inversion characteristic of high ozone-concentration periods; the base of layer 3 is typically 1 to 2 km above the ground. Relatively clean tropospheric air is assumed to exist above layer 3 at all times. If cumulus clouds are present, an upward flux of ozone and precursor species is injected into the layer by penetrative convection. At night, ozone and the remnants of other photochemical reaction products may remain in this layer and be transported long distances downwind. These processes are modeled in layer 3.

When cumulus clouds are present in a layer 3 cell, the upward vertical mass flux from the surface is partially diverted from injection into layer 1 to injection directly into the cumulus cloud of layer 3. In the atmosphere, strong thermal vertical updrafts, primarily originating near the surface in the lowest portion of the mixed layer, feed growing fair weather cumulus clouds with vertical air currents that extend in one steady upward motion from the ground to well above the top of the mixed layer. These types of clouds are termed *fair weather cumulus* since atmospheric conditions are such that they do not grow to the extent that precipitation forms. The dynamic effects of this transport process and daytime cloud evolution can have significant effects on the chemical fate of pollutants. For example, fresh emissions from the surface layer can be injected into a warm thermal and rise, essentially unmixed, to the top of the mixing layer where they enter the base of a growing cumulus cloud. Within the cloud, the chemical processes of ambient pollutant species are suddenly altered by the presence of liquid water and the attendant attenuation of sunlight. The presence of fair weather cumulus clouds implies that the atmosphere above the earth's boundary layer is too stably stratified for thermals to penetrate higher. In this case, the air comprising the tops of these clouds returns to the mixed layer and is heated on its descent since it is being compressed by increasing atmospheric pressures. Ultimately, the air again arrives at the surface level where new emissions can be injected into it and ground deposition may occur, and the process may begin again. The time required for one complete cycle is typically 30 to 50 minutes with perhaps one-tenth of the time spent in the cloud stage.

Within the ROM system, a submodel parameterizes the above cloud flux process and its impact on mass fluxes among all the model's layers. In the current implementation of the chemical kinetics, liquid-phase chemistry is not modeled, and thus part of the effects from the cloud flux processes are not accounted for in the simulations. Future versions of the chemical kinetics may include liquid-phase reactions. The magnitude of the mass flux proceeding directly from the surface layer to the cloud layer is modeled as being proportional to the observed amount of cumulus cloud coverage and inversely proportional to the observed depth of the clouds.

Horizontal transport within the ROM system is governed by hourly wind fields that are interpolated from periodic wind observations made from upper-air soundings and surface measurements. During the nighttime simulation period, the lowest few hundred meters of the atmosphere above the ground may become stable as a radiation inversion forms. Wind speeds increase just above the top of this layer, forming the nocturnal jet. This jet is capable of carrying ozone, other reaction products, and emissions injected aloft considerable distances downwind. This phenomenon is potentially significant in modeling regional-scale air quality and is implicitly treated by the model, where the definition of layer 1 attempts to account for it.

Because standard weather observations do not have the spatial or temporal resolution necessary to determine with confidence the wind fields in layer 1, a submodel within the ROM system was developed to simulate the nighttime flow regime in layer 1 only. This prognostic flow submodel is activated only when a surface inversion is present over most of the model domain. At all other times, the flow in layer 1 is determined from interpolation of observed winds. The nighttime flow regime within layer 1 is influenced by buoyancy, terrain, warm cities, pressure-gradient forcing, and frictional forces, all of which are accounted for in the model's flow formulation. Solution of the wind submodel equations produces estimates of the wind components as well as the depth of the inversion layer for all grid cells in layer 1.

1.2.2 ROM Chemistry

The chemical kinetic mechanism embedded in the current version of the ROM is the Carbon Bond IV (CB-IV) set of reactions (Gery *et al.*, 1989). This mechanism simulates the significant reaction pathways responsible for gas-phase production and destruction of the constituents of photochemical smog on regional scales. The mechanism consists of 82 reactions encompassing 35 individual species; these species are listed in Table 1. The ROM's chemical solution scheme makes no *a priori* assumptions concerning local steady states. Therefore, all species are advected, diffused, and chemically reacted in the model simulations.

The CB-IV contains a standard set of reactions for atmospheric inorganic chemical species, including O_3, NO, NO_2, CO, and other intermediate and radical species. Organic chemistry is partitioned along reactivity lines based on the carbon structures of the organic molecules. Nine individual categories of organics are represented to account for the chemistry of the hundreds of organic molecules existing in the ambient atmosphere: ETH, an explicit representation of ethene; FORM, an explicit representation of formaldehyde; OLE, a double-bonded lumped structure including two carbons (e.g., olefins); PAR, a single-bond, single-carbon structure (i.e., paraffins); ALD2, the oxygenated two-carbon structure of the higher aldehydes; TOL, the aromatic structure of molecules with only one functional group (e.g., toluene); XYL, the structure of molecules with multifunctional aromatic rings (e.g., xylene); ISOP, the five-carbon isoprene molecule; and NONR, a single-carbon organic structure not significantly participating in the reaction sequence. We include MTHL (methanol) in the mechanism for future-year scenarios that require emission control strategies for methanol-powered vehicles.

TABLE 1. ROM CHEMICAL SPECIES [a]

Symbol	Description	Symbol	Description
ALD2	High MW aldehydes	O1D	O^1D atom
C2O3	Peroxyacetyl radical	O3	Ozone
CO	Carbon monoxide	OH	Hydroxyl radical
CRES	Cresol and high MW phenols	OLE	Olefinic carbon bond
CRO	Methylphenoxy radical	OPEN	High MW aromatic oxidation ring fragment
ETH	Ethene		
FORM	Formaldehyde	PAN	Peroxyacetyl nitrate
H2O2	Hydrogen peroxide	PAR	Paraffinic carbon bond
HNO2	Nitrous acid	PNA	Peroxynitric acid
HNO3	Nitric acid	ROR	Secondary organic oxy radical
HO2	Hydroperoxy radical	TO2	Toluene-hydroxyl radical adduct
ISOP	Isoprene structures	TOL	Toluene
MGLY	Methylglyoxal	XO2	NO to NO_2 reaction
N2O5	Dinitrogen pentoxide	XO2N	NO to nitrate (NO_3^-) reaction
NO	Nitric oxide	XYL	Xylene
NO2	Nitrogen dioxide	MTHL	Methanol
NO3	Nitrogen trioxide	NONR	Nonreactive hydrocarbons
O	O^3P atom	TRAC	Tracer species

[a] MW = molecular weight

Three classes of biogenic hydrocarbons are included in a separate natural area source emissions inventory used by the ROM: (1) isoprene, a molecule principally emitted by deciduous trees, is treated by the ISOP species in CB-IV; (2) monoterpenes, a class of natural hydrocarbons emitted principally by coniferous trees, is not treated explicitly in CB-IV. The surrogate monoterpene molecule, α-pinene, which consists of 10 carbons, is apportioned to the existing CB-IV categories as 0.5 OLE, 1.5 ALD2, and 6 PAR; (3) unidentified hydrocarbons (gas chromatography analysis did not identify specific hydrocarbon compounds) are tentatively treated as 50% terpenes, 45% PAR, and 5% NONR (Pierce *et al.*, 1990). These unidentified compounds can comprise as much as 40% of the biogenic hydrocarbons.

1.2.3 System Components

The raw input data to the ROM (refer to Section 1.3) are manipulated by a hierarchical network of processors that range in function from simple reformatting of emissions data to generating the complex wind fields that drive the atmospheric transport algorithm in the Core Model. These processors are interconnected by their requirements for and production of data. The ultimate product of the processor network is a collection of data files that can be categorized into two types: processor files (PF) and model files (MF). Processor files contain partially processed data required as input to higher level processors. Model files contain the parameter fields that are transformed into the variables required by the model algorithms; however, they also provide input to a number of higher level processors. The output of the processors are the four Core Model input data files. The Core Model is described in detail starting with Section 1.5. Figure 3 shows the principal components of the model, starting with the raw input data and ending with the CONC file.

The processors are organized into nine distinct hierarchical stages, numbered 0 - 8. Stage 0 processors produce output files such as the gridded land use data. Stage 1 processors interface directly with the preprocessed input data sets, which have, at this juncture, undergone extensive quality control. Subsequent stages transform the input data into the gridded fields of temporally and spatially varying parameter values needed by the highest stages of the processing network. Processors at any stage can interface directly with the B-Matrix compiler, described below, by production of model input files (MF). This multistage organization is important to the network because it clearly delineates the sequence of program execution. Processors at the same stage may execute simultaneously. A processor at any given stage, however, must wait until all processors from lower stages along its input data paths have been completed. Formal definition of all data/processor relationships and automation of processor executions are essential to ensure consistency and validity of model input files.

The program that serves as the interface between the model input files and the algorithms describing the governing processes is called the B-Matrix Compiler (BMC) because it functions similarly to a computer language compiler that transforms high-level language commands into a machine or algorithm-specific representation. The BMC mathematically combines physical parameters such as layer thicknesses, air densities, etc., into the complex coefficients required for solution of the governing equations. These coefficients can no longer be equated with physical quantities; they are purely mathematical entities related specifically to the form of the finite difference algorithms used by the ROM.

The core of the ROM system is a set of algorithms that solves the coupled set of finite difference equations describing the governing processes in each layer of the model. These governing equations are expressed in a form that allows the chemical kinetics, advection, and vertical flux to be treated independently. The chemistry module exchanges information with algorithms of the governing equations via two vectors: (1) a vector that contains the net production rate of each species, and (2) a vector that contains the net destruction rate. Such design simplifications enhance the flexibility of the model and are not limited to the interchanges of the chemical mechanism; they apply to all theoretical formulations of the physical and meteorological processes (i.e., to all the processors).

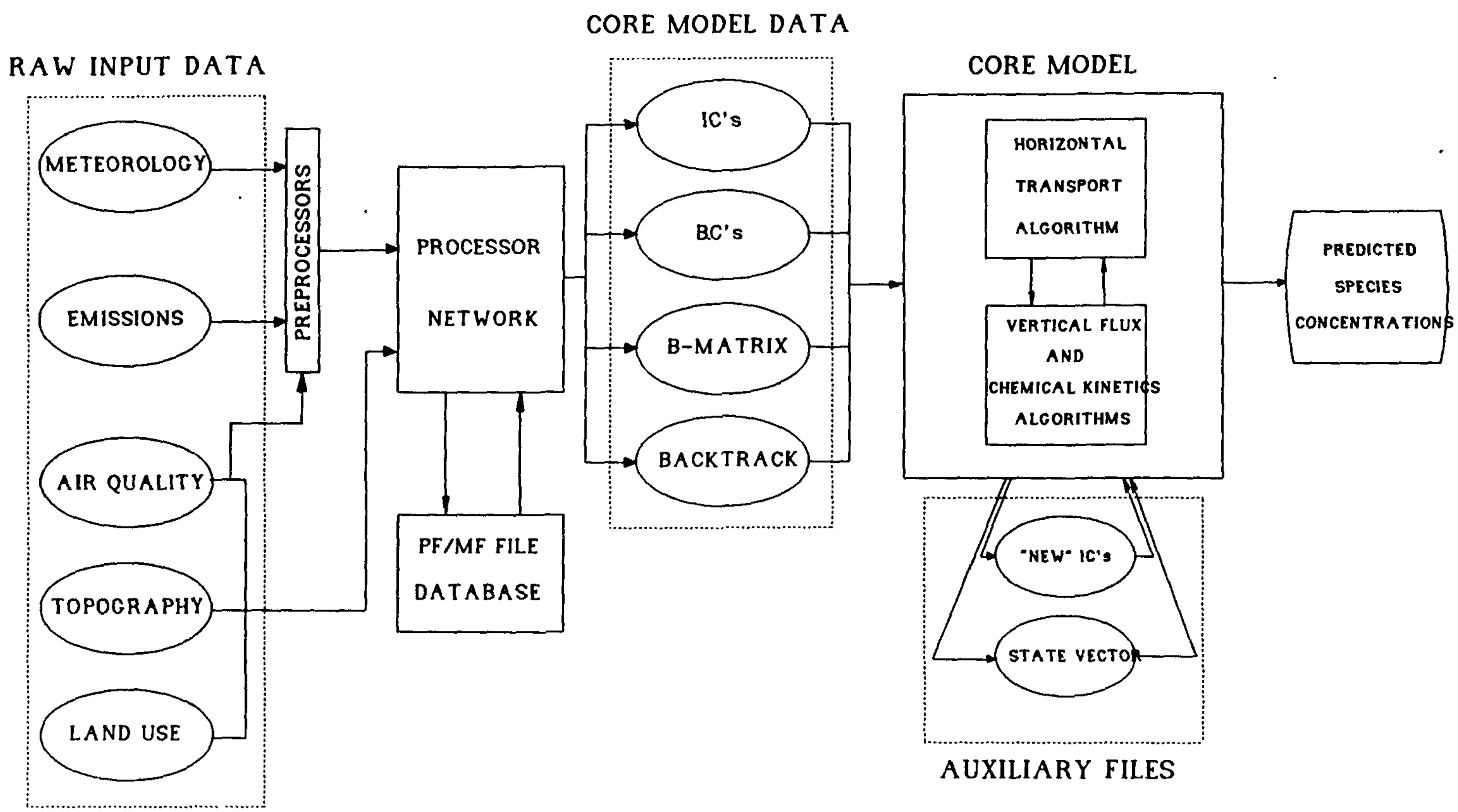


Figure 2.2. Components of the ROM.

1.2.4 ROM Limitations

There are several limitations inherent in the model. Among the most important of these are: (1) the model is designed to represent only ozone-season (April - October) meteorological conditions; (2) the ROM does not take into account any aqueous-phase chemistry; (3) cumulus cloud processes are such that when a cloud is created in a grid cell, it remains there for a full hour (i.e. cloud physics are not considered), and the cloud is not advected; (4) the ROM actually consists of three two-dimensional models that are linked,[4] and, as such, cannot be expected to model regions that contain high, complex, mountainous terrain such as the Rocky Mountains or Sierra Nevada; (5) the ROM, with its current 18.5 km x 18.5 km grid resolution, is not designed to provide detailed information at local scales that are significantly influenced by local source distributions; and (8) the ROM is currently configured to run only on domains in eastern North America on the scale of 1000 km. This regional scale is germane since long-range transport of ozone precursors will have a significant effect on local ozone concentrations.

We advise users of ROM to interpret its results in terms of analysis of different emission control strategies on ozone concentration, rather than assuming the results to be an accurate snapshot of a specific pollution event.

1.3 DATA AND FILE REQUIREMENTS

The ROM system requires five types of "raw" data inputs: air quality, meteorology, emissions, land use, and topography. Air quality data required by the ROM include initial conditions (IC) and boundary conditions (BC). The model is initialized with clean tropospheric conditions for all species several (usually 2-4) days before the start of an episode. The initial condition field has essentially been transported out of the model domain in advance of the portion of the episode of greatest interest. Upwind lateral boundary conditions for ozone are updated every 12 hours based on measurements. Other species concentrations at the boundaries, as well as all species at the top of the modeling domain, are set to tropospheric clean-air concentrations.

Meteorological data are assimilated by the first stage of preprocessors. These data contain regular hourly observations from U.S. National Weather Service surface stations (and from similar stations in Canada as necessary), including wind speed and direction, air temperature and dew point, atmospheric pressure, and cloud amounts and heights. Twice-daily sounding data from the upper-air observation network are also included in the meteorological database. Upper-air meteorological parameters include atmospheric

4. The three models represent layer 3, layer 2, and a combined layer 1 and 0.

pressure, wind speed and direction, and air temperature and dew point. Finally, both buoy and Coastal Marine Automated Station data are used; parameters typically reported are wind speed and direction, and air and sea temperatures.

Emissions data for the primary species are input to the ROM system as well. Most recently these data have been provided from the NAPAP 1985 emissions inventory with 20-km spatial resolution (Saeger *et al.*, 1989). Species included are CO, NO, NO_2, and nine hydrocarbon reactivity categories. Natural hydrocarbons are also input, including isoprene explicitly, monoterpenes divided among the existing reactivity classes, and unidentified hydrocarbons.

The Core Model requires eight input files and one output file to exist in order to run; these files are summarized below. Each of the six principal input files starts with a text file header that describes the file's contents, followed by a data body. The data body consists of a sequence of time step headers, each followed by a body of data records relative to that time step. While these files all contain data, note that the auxiliary input files PROG and STOPCK do not.

Principal files

- BCON - The boundary conditions file contains the concentrations for each of the 35 chemical species in a one-cell deep border *within* the modeling domain. Concentrations are given for each model layer and for each model time step. We show the file's structure diagram in Appendix B, Figure B-1.
- BMAT - The B-matrix file contains (1) the vertical flux parameters that are required to solve the B-matrix differential equation, and (2) the emission source terms, and (3) the gridded meteorology parameters in the three model layers for each model time step (1800 s) necessary to make adjustments in chemistry rate constants. We show the file's structure diagram in Appendix B, Figure B-2.
- BTRK - The Backtrack file contains the gridded locations (in grid cell coordinates) of the previous position of a parcel of air that will arrive at a grid cell node at the current time step. These data are used by the model to compute the horizontal transport. We show the file's structure diagram in Appendix B, Figure B-3.
- ICON - The initial conditions file contains the gridded concentrations for the 35 chemical species in each model layer for the initial model time step. This file is used only at the start of an episode time-contiguous run, i.e., at the start of scenario 1. It is supplanted by the NEW-ICON file at the start of each subsequent scenario within the episode.[5] We show the file's structure diagram in Appendix B, Figure B-4.

5. The relationship between *scenarios* and *episodes* is that each *scenario* is 72 hours long, and an ozone *episode* consists of one or more contiguous *scenarios*.

- NEWICON - The model writes this file at the completion of the last time step of a scenario. It is used as the ICON file for the continuation of the episode computation, and is a copy of the CONC file from the last time step of the scenario. NEWICON supplants the ICON file for the continuation of the episode computation. The CONC file itself could be used to continue the episode, but the NEWICON file is much smaller; thus, continuing runs do not require retaining access to the large CONC files.

- RESTRT - This file constitutes a *state vector* file, and is used only to restart a model run in the event of program termination before the end of a scenario. RESTRT contains the data that correctly reposition the input files and set the text pointers in the code; this ensures the proper state of the model when you resume a run from the start of any scenario time step prior to the termination step.

Auxiliary files

- PROG - This file is a one line text file that the model writes at the completion of each time step. Its purpose is to allow you to view the progress of the model; you can "type" out this file during the model run and see the number of the last completed time step.
- STOPCK - The purpose of this file is to allow you to terminate a model run at the end of a currently-executing time step. It consists of one line of text enclosed in single quotes, which, if anything other than 'STOP', permits the model run to continue.

Output file

- CONC - The model writes the predicted concentrations that result from its execution to the CONC file. The data consist of the gridded concentrations of the 35 chemical species within each of the model layers for each time step (30 minutes) of the model run. We show the file's structure diagram in Appendix B, Figure B-5.

1.4 MODEL EXECUTION AND STORAGE REQUIREMENTS FOR THE IBM 3090

1.4.1 Starting the ROM at the Top of an Hour

If you want to run consecutive scenarios, the Core Model <u>must</u> be started at the top of an hour, not at the half-hour. We begin all our model runs at noon EST.

1.4.2 CPU Use

Currently, we are running the ROM Core Model on the EPA's IBM® 3090 computer.[6] Running the ROM requires significant CPU resources; a typical 3-day simulation (72 hours, 144 model time steps) requires about 9.5 hours of CPU time on the IBM 3090. Approximately 10 percent of this CPU time

6. IBM is a registered trademark of the International Business Machines Corporation.

is used in the calculation of advection; the other 90 percent is used for the gas-phase chemistry calculations. With the parallel processing capability of the IBM 3090, elapsed clock time is approximately one-quarter of the CPU time.

1.4.3 Storage Requirements

A list of storage requirements for the Core Model follows (1 track = 47,476 bytes):

- 21 tracks of code (approximately 12000 lines of code)
- 24 tracks of libraries
- 1 track of compile/link JCL
- 1 track of JCL to submit the run
- 47 tracks of the log file

1.4.4 Core Model Input Data: Transfer from the VAX to the IBM

The following procedure is specific to our installation, and may not pertain to you. The Core Model input data sets are transferred from the VAX to the IBM using the DECnet/SNA Data Transfer Facility™ (DTF).[7] For processors that generate Core Model input data sets,[8] you will select a flag in the control cards file that determines whether to write the data in VAX binary or IBM binary format. We select the IBM format, and invoke the DTF to directly write the ICON, BCON, BTRK, and BMAT files to the IBM's disks. For further information on transferring data, we refer you to the DECnet/SNA VMS Data Transfer Facility User's Guide.[9] The subroutine that translates data to IBM-readable format is briefly described below.

SUBROUTINE TRNSLT

This subroutine translates VAX format and writes an IBM-format file on VAX Direct Access Storage Drive (DASD) so that it may subsequently be copied to IBM DASD via the DEC/DTF (Data Transfer Facility). The translation is necessary because DTF supports only a one-to-one transfer of the data, which do not map to identical representation of numbers on the two architectures. If the file to be transferred is ASCII in its entirety, a call to this routine is unnecessary, and the DTF facility can be used directly. If, however, the file is a combination of ASCII and REAL*4 or INTEGER*4 data, a

7. DECnet is a trademark of Digital Equipment Corporation, Nashua, NH 03061.
8. P02G, P22G, P38G, and P40G.
9. Order number AA-JM75B-TE; Digital Equipment Corporation, P.O. Box CS2008, Nashua, NH 03061.

call to this routine is necessary. This routine does not support any other VAX data structures. TRNSLT translates VAX F_floating format words to IBM floating point, and ASCII format to EBCDIC format.

1.5 SOFTWARE COMPONENTS OF THE CORE MODEL

Figure 4 shows a conceptual data-flow diagram of the Core Model; its design reflects our desire to keep the model as simple as possible so that we can readily maintain and upgrade it. The design is based on the concept of modularity with respect to the flow of data, and follows the ideas of Jackson (see Appendix A). The system specification diagram that illustrates the main data flows is shown in Appendix C, Figure C-1. Most of the subroutines in the Core Model were written as though they were stand-alone programs that read and write intermediate files during the computation. Following Jackson, these subroutines were then "inverted," and the intermediate files eliminated. An illustration of our implementation of this Jackson Structured Design for the Core Model can be found in Appendix C, Figure C-2.

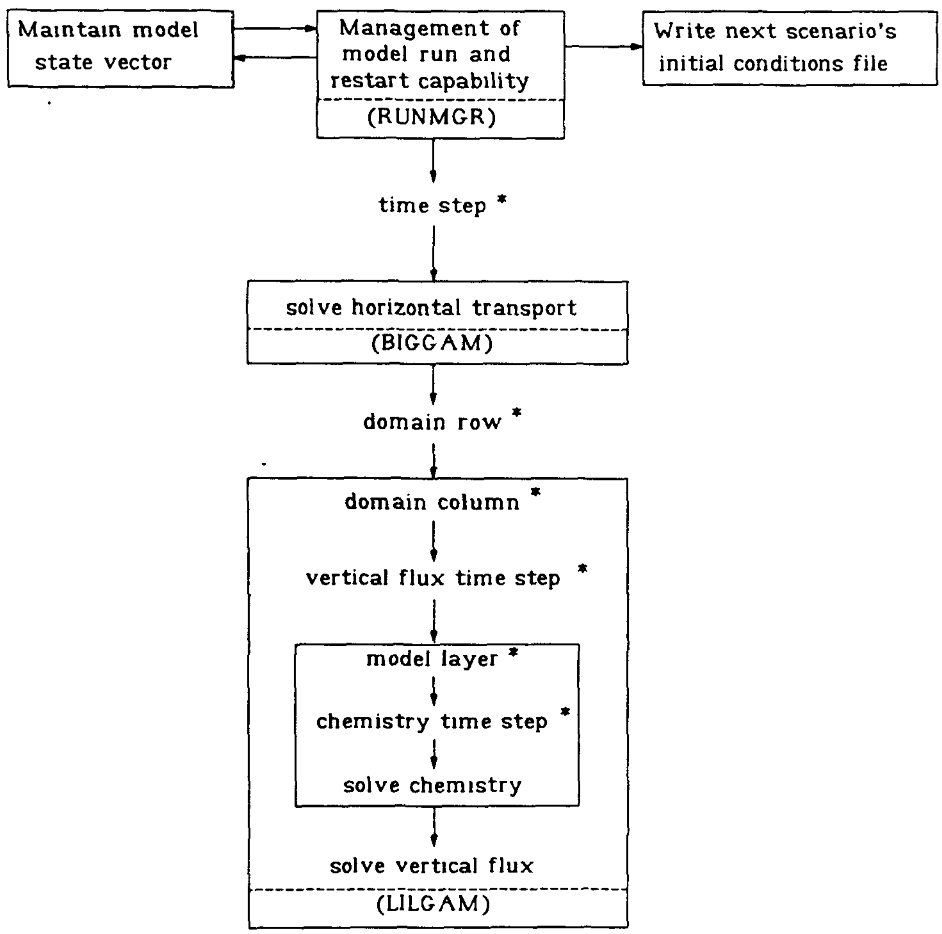


Figure 4. Conceptual data-flow diagram of the Core Model. Asterisks indicate iterative loops.

The Core Model consists of three principal components: RUNMGR, BIGGAM, and LILGAM. The main program is RUNMGR (Run Manager); its purpose is to: (1) initiate the run, (2) restart the run if required, (3) maintain the state vector file (RESTRT), (4) write the progress file (PROG), and (5) examine the stop check file (STOPCK). Once the required input files are properly opened, RUNMGR invokes BIGGAM, starting the actual computation process. BIGGAM in turn invokes LILGAM. Because of the large time-scale differences between the transport processes and the chemical processes in the regional-scale oxidant modeling problem, we can write the diffusion equation as two processes to solve for a transport component Γ decoupled from the chemistry. Then, with the solution for Γ in hand, we can determine the chemistry component γ. We show the structure diagram for RUNMGR in Appendix C, Figure C-3.

1.5.1 BIGGAM

BIGGAM solves for the horizontal advection of the chemical species concentrations. The transport is computed by (1) interpolating the concentration values at backtrack locations,[10] and (2) solving the transport using a quasi-Lagrangian advection scheme that involves determining a Green's function for the Γ diffusion equation at spatially-interpolated upwind points. We use a biquintic surface fit for the spatial interpolation. For a one-cell deep perimeter within the modeling domain boundaries, BCON file concentrations supplant the interpolated concentrations if inflow conditions exist at that border of the domain. Once BIGGAM has computed an advection solution component, it invokes LILGAM to compute the chemistry component and finalize the current time step concentration predictions. We show the structure diagram for BIGGAM in Appendix C, Figure C-4.

The subprograms specifically associated with BIGGAM are shown in Figure 5, and then are briefly described below.

INIRUN reads the run stream's control parameter cards and sets up the model run conditions. INIRUN loads the common blocks in the include file HEADIN.EXT and performs the following operations:

- Sets the vertical flux time step (300 s) and the chemistry solver parameters, and echoes these data to the run log.
- Sets the number (3) and names (O_3, NO, NO_2) of the primary oxidant species in the ROM, and reports these data to the run log.
- Obtains the indices of the primary oxidant species using the utility function INDEX1, and reports these data to the run log.
- Sets and echoes the upper and lower chemistry time step limits.
- Sets the advection time step variable used in LILGAM equal to the corresponding variable in HEADIN.EXT.

10. A backtrack location is the prior position of a parcel of air that will arrive at a grid cell node at the current time step, where each time step is 1800 s.

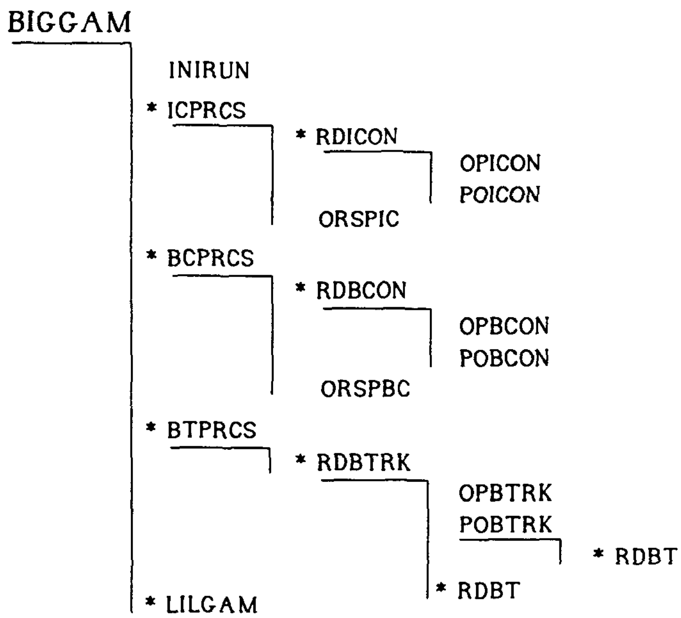


Figure 5. BIGGAM subprograms. An asterisk implies that the subprogram can be called more than one time.

- Calls ADATE to obtain and report the execution creation date and time to the run log.
- Initializes all the file date and time information to correspond to the model time initialization. These data are used to maintain all the process subroutine's internal clocks.

The variables in the common blocks contained in HEADIN.EXT are used throughout the model to:

- Set upper limits for processing loops,
- verify file headers on the input files, and
- provide data to be written to the CONC file header.

For a complete description of these variables and their origin, see Table 2.

ICPRCS is the process interface between BIGGAM and the time step initial conditions data, restructured for the model data processing; its structure diagram is shown in Appendix C, Figure C-5.

RDICON is the process interface between ICPRCS and the initial conditions data in the ICON or the NEWICON file; its structure diagram is shown in Appendix C, Figure C-6.

OPICON opens the ICON or the NEWICON file, reads and checks the file's header information, then writes a summary to the run log.

TABLE 2. HEADIN.EXT VARIABLES: DESCRIPTIONS AND ORIGINS

Name	Description	Origin
	DATE/TIME INFORMATION	
CDATIN	Creation date of simulation (*YYMMDD*)	Argument returned from call to DATTIM
CTIMIN	Creation time of simulation (*HHMMSS*)	Argument returned from call to DATTIM
SDATIN	Julian start date for model scenario (*YYDDD*)	Read in from run stream
STHRIN	Start hour (00 - 23) for model scenario	Read in from run stream
TSTPIN	Time step size (s) for model scenario	Set in assignment statement equal to 1800
FRSTIN	First time step (s) past start time of model scenario	Argument returned from call to CLOCK1
	GRID DEFINITION INFORMATION	
GRDNIN	Grid definition name	Read in from run stream
SWLNIN	Longitude (degrees) of southwest corner of grid	Argument returned from call to CELLM
SWLTIN	Latitude (degrees) of southwest corner of grid	Argument returned from call to CELLM
NELNIN	Longitude (degrees) of northeast corner of grid	Argument returned from call to CELLM
NELTIN	Latitude (degrees) of northeast corner of grid	Argument returned from call to CELLM
DLONIN	Longitudinal grid cell increment (degrees)	Argument returned from call to CELLM
DLATIN	Latitudinal grid cell increment (degrees)	Argument returned from call to CELLM
NCOLIN	Number of columns in domain grid	Argument returned from call to CELLM
NROWIN	Number of rows in domain grid	Argument returned from call to CELLM
	MODEL LAYERS INFORMATION	
NLEVIN	Number of model layers	Set in assignment statement, equal to NLEVS (DIMENS parameter)
LVNMIN(l)	Name of model layer *l*	Set in assignment statement, equal to LVNAME (BLKMOD data)
	MODEL SPECIES INFORMATION	
NSPCIN	Number of chemical species in model	Set in assignment statement, equal to NSPECS (DIMENS parameter)
SPNMIN(k)	Name of species *k*	Set in assignment statement, equal to SPNAME (BLKMOD data)
	CONC FILE HEADER DESCRIPTIVE TEXT	
ICNTIN	Number of text records	Computed during the operation that reads in the text records
TEXTIN(n)	Descriptive text record *n*	Read in from run steam

POICON positions the ICON or the NEWICON file to the start of the model execution scenario.

ORSPIC generates the expansion list (look-up table) of species names for which the ICON file has values. These values are mapped to the full Core Model list of species. For ROM2.1, the ICON file contains the same species list as the Core Model, and thus the mapping table is one-to-one.

BCPRCS is the process interface between BIGGAM and the boundary conditions data, restructured for the model data processing; its structure diagram is shown in Appendix C, Figure C-7.

RDBCON is the process interface between BCPRCS and the boundary conditions data in the BCON file; its structure diagram is shown in Appendix C, Figure C-8.

OPBCON opens the BCON file, reads and checks the file's header information, then writes a summary to the run log.

POBCON positions the BCON file to the start of the model execution scenario.

ORSPBC generates the expansion list (look-up table) of species names for which the BCON file has values. These values are mapped to the full Core Model list of species. For ROM2.1, the BCON file contains the same species list as the Core Model, and thus the mapping table is one-to-one.

BTPRCS is the process interface between BIGGAM and the backtrack data in the BTRK file, reorganized for BIGGAM data processing; its structure diagram is shown in Appendix C, Figure C-9.

RDBTRK is the process interface between BTPRCS and the data in the BTRK file; its structure diagram is shown in Appendix C, Figure C-10.

OPBTRK opens the BTRK file, reads and checks the file's header information, then writes a summary to the run log.

POBTRK positions the BTRK file to the start of an execution scenario.

RDBT performs the FORTRAN open and the file and data read operations on the BTRK file; its structure diagram is shown in Appendix C, Figure C-11.

1.5.2 LILGAM

LILGAM computes the remaining components of the species concentrations: those due to vertical fluxes across the model layer surfaces, $\bar{\gamma}$, and those due to the chemical reactions between the pollutant species, γ'. The value of the concentration recorded at the end of each 1800-sec advection time step is the product

$$C = \Gamma \cdot \tilde{\gamma} \cdot \gamma'$$

- Every 300 s within the advection time step, the interlayer vertical fluxes are modeled using a fourth-order Runge-Kutta integration of the system of differential equations that describes them. The integration step size is determined by finding the largest negative eigenvalue of the third-order matrix of coefficients (B-matrix) that quantifies the interlayer fluxes. The reciprocal of this eigenvalue is proportional to the smallest time scale in the system describing the vertical exchange between the model's three layers.
- Within each 300 s $\tilde{\gamma}$ time step, the chemistry component γ' is computed using a small time step that varies between 10 s and 60 s. The differential equations that describe the chemical reactions are solved iteratively, and model the equilibration process of a reactive mixture of gaseous pollutant species. Essentially, the time step is controlled by the stiffness of the system of differential equations, which in turn is a function mainly of local primary emissions and sunlight amount. The relative stiffness of the system is determined by comparing (at the end of each chemistry time step) the fractional difference in the nitric oxide

(NO) species from the previous step and adjusting the time step downward if the difference exceeds a preset value. NO is used as the standard since it is both one of the primary oxidant species and is involved in the fastest reactions in the system.

We show the structure diagram for LILGAM in Appendix C, Figure C-12. The subprograms specifically associated with LILGAM are shown in Figure 6, and then briefly described.

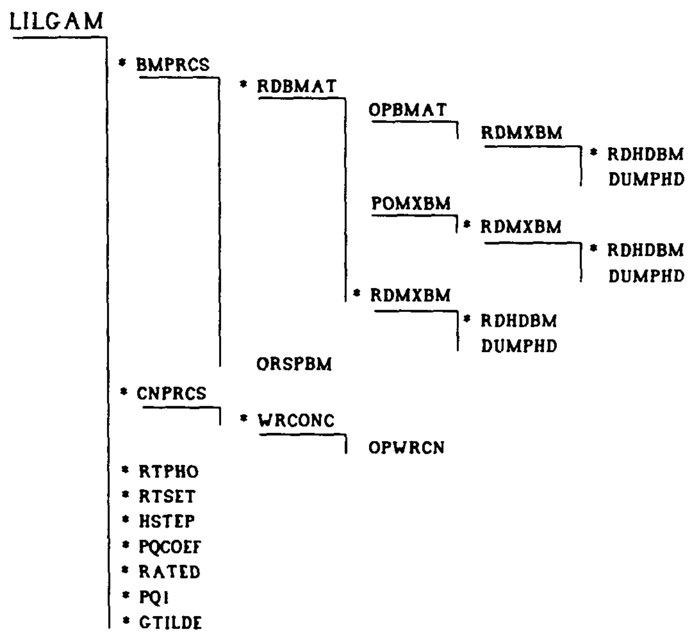


Figure 6. LILGAM subprograms. An asterisk implies that the subprogram can be called more than one time.

BMPRCS is the process interface between LILGAM and the BMAT file data, reorganized for the LILGAM data processing; its structure diagram is shown in Appendix C, Figure C-13.

RDBMAT is the process interface between BMPRCS and the data on the BMAT file; its structure diagram is shown in Appendix C, Figure C-14.

OPBMAT opens the BMAT file, reads and checks the file's header information, then writes a summary to the run log.

RDMXBM performs the FORTRAN open and the file and data read operations on the (possibly multiple) BMAT file; its structure diagram is shown in Appendix C, Figure C-15.

RDHDBM reads the BMAT file header on each of the BMAT's subfiles and echoes the information to the run log.

DUMPHD is a subroutine invoked by RDMXBM to dump the BMAT file header information to the run log if multiple BMAT files are not correctly ordered.

POMXBM positions the (possibly multiple) BMAT file to the start of the execution scenario.

ORSPBM generates the expansion list (look-up table) of species names for which the BMAT file has values. These values are mapped to the full Core Model list of species.

CNPRCS is the process interface between LILGAM and the predicted concentration data, reorganized for writing to the CONC file; its structure diagram is shown in Appendix C, Figure C-16.

WRCONC is the process interface between CNPRCS and the concentrations data in the CONC file; its structure diagram is shown in Appendix C, Figure C-17.

OPWRCN opens the CONC file, writes the file's header information on the file, and echoes this data to the run log.

RTPHO determines the layer average photolytic rate constants, adjusted for variations in solar radiation due to cloud cover and solar angle.

RTSET adjusts the values of rate constants for the nonphotolytic reactions for temperature, air density, and water vapor concentration.

HSTEP determines the integration step size used in GTILDE.

PQCOEF calculates the part of the normalized reaction coefficients that do not depend on the current chemistry component of the concentration, γ'.

RATED completes the chemistry time step calculation of the reaction coefficients using the current value of γ'.

PQ1 calculates the production and decay terms of the chemistry differential equation system for each chemistry time step. LILGAM then uses these terms in a predictor/corrector fashion to update γ' for the chemistry time step.

GTILDE calculates the interlayer flux concentration component, $\tilde{\gamma}$, for each of layer 1, layer 2, and layer 3.

1.6 SUBPROGRAMS THAT MAINTAIN THE STATE VECTOR AND RESTART THE MODEL

The subprograms specifically associated with maintaining the state vector file and with model run restarts are listed below, then briefly described.

RDCONC
OPCONC
POCONC
WRSTAV
RDSTAV
OPSTAV
POSTAV

RDCONC is the process interface that is called for a RESTART run; it reads the CONC file data for the time step that precedes the first step of the restarted run, and its structure diagram is shown in Appendix C, Figure C-18.

OPCONC opens the CONC file for a RESTART run, reads and checks the file's header information, then writes a summary to the run log.

POCONC positions the CONC file, for a RESTART run, to the requested starting time.

WRSTAV opens and writes the state vector data to the RESTRT file.

RDSTAV reads the data on the RESTRT file for a RESTART run, for the time step that precedes the first step of the restarted run.

OPSTAV opens the RESTRT file for a RESTART run.

POSTAV positions the RESTRT file for a RESTART run to the requested starting time.

1.7 UTILITY SUBPROGRAMS

ADATE
ASORT
CELLM
CLOCK1
CLOCK2
DATTIM
FSKIP1
INDEX1
IOCL
JFILE2
JFILE5
JFILE6
JULIAN
JUNIT
RDCHAR

RDFILE
WRCHAR
WRFILE

ADATE returns the current formatted date and time from the computer operating system by calling the FORTRAN library subroutines IDATE and TIME.

ASORT sorts an array of character names into alphabetical order.

CELLM returns the Northeast and Southwest corner latitudes and longitudes of one of three regions (NEROS1, SEROS1, or ROMNET1). It also returns the number of columns and rows in the grid domain and the grid cell latitude and longitude increments.

CLOCK1 returns the time elapsed from the scenario start, and the current time step number.

CLOCK2 returns the current date and time, and the time elapsed from the scenario start.

DATTIM returns a formatted current date and time using subroutine ADATE.

FSKIP1 positions formatted or unformatted sequential files.

INDEX1 is a function subprogram that returns the position of a character name in a list of names.

IOCL is a function subprogram that returns a clause field in an I/O status word. In the Core Model, this field is reported to the run log in case of an I/O error to assist you in determining the cause of the error.

JFILE2 is a function subprogram that opens a sequential file and returns the FORTRAN unit number to which it is attached.

JFILE5 is a function subprogram that opens a sequential file with fixed length records and returns the FORTRAN unit number to which it is attached.

JFILE6 is a function subprogram that opens a sequential file with FORTRAN carriage control and variable record type to enable the DEC Data Transfer Facility (DTF) to pass binary VAX records to the IBM. JFILE6 returns the FORTRAN unit number to which the file is attached. This subprogram is specific to our installation, and may not pertain to you.

JULIAN is a function subprogram that returns a given calendar year, month and day as the Julian date *YYDDD*.

JUNIT is a function subprogram that keeps track of the FORTRAN I/O units already assigned in the model and returns the next available unit number.

RDCHAR performs an unformatted read into a character buffer whose length is a passed argument.

RDFILE performs an unformatted read into a numeric buffer whose length is a passed argument.

WRCHAR performs an unformatted write from a character buffer of arbitrary length.

WRFILE performs an unformatted write from a numeric buffer of arbitrary length.

1.8 MISCELLANEOUS SUBPROGRAMS

BLKMOD
NEWICS
PRGSMY
TIMER
CPUTIM

BLKMOD contains the FORTRAN BLOCK DATA initializations for:

- the standard read and write I/O units,
- the process subroutine text pointers,
- the global formats that are used to echo file header information to the run log when the files have been successfully opened,
- the internal (logical) names for all the model files,
- the 35 chemical species names used in ROM2.1,
- the 3 model layer names,
- the STOPCK file's flag initial value,
- the concentration species values used to represent zero in the model, i.e., 10^{-16} ppm.
- the values for the basic rate constants,
- the 11 photolytic rate constant levels used by RTPHO to determine the integral average photolytic rate constants,
- the arrays of clear sky photolytic rate constants tabulated for the five CBM4.2 photolytic reactions, the solar zenith angle, and the 11 photolytic rate constant levels.

NEWICS writes a copy of the CONC file's concentration data to the NEWICON file at the end of a model execution scenario.

PRGSMY records summary information of all the subprograms used to create the executable image of the model on the run log.

TIMER determines elapsed CPU and elapsed clock time using subroutine CPUTIM.

CPUTIM retrieves from a VAX system the CPU time elapsed since its initial call during a program execution.

1.9 CORE MODEL OUTPUT DATA: TRANSFER FROM THE IBM TO THE VAX

The following procedure is specific to our installation, and may not pertain to you. After the model run completes, we translate the CONC file to VAX binary format and send the file to the VAX through the DTF. The program that does the translation is CNTRAN.FOR, which also verifies that all the time steps for a run are present and backs up the untranslated file on the IBM. The data are translated to ASCII and VAX floating point format. We show the JCL for this program below.

```
//XHST87SA JOB (NER1RSMRP,B132,,,,,,58),'HALLYBURTON',MSGCLASS=P,      00010099
//            NOTIFY=XHS,TIME=(50,),PRTY=2                             00031099
/*AFTER XHSR87SA                                                       00040099
/*JOBPARM LINES=999                                                    00050099
/*ROUTE PRINT RMT378                                                   00060099
//PROCLIB DD DSN=XHSNER1.ROMNET1.PROCLIB,DISP=SHR
//*
//* This procedure will translate, verify, and backup
//* a ROMNET production conc file.  The translated file will have the
//* same name as the original + .TRANS, the tape backup file will be
//* named ADRNER1.CONC.filnam.
//*
//* If this is NOT what you want...do NOT use this procedure
//*
//*
//TVBGCN EXEC TVBGCN,REGION=0K,COND=(0,NE),
//TOPNAM='ADRNER1.NOBKUP.CONC.',FILNAM='FV187SA'
//*
//*****************************************************************
// EXEC SAS,GREGION=6000K,OPTIONS=MACRO,COND=(0,NE),PRINT='*'
//SAS.WORK DD SPACE=(CYL,(300,20)),VOL=(,,,15)
//IN       DD DSN=*.TVBGCN.CTRN1.CONDISK,DISP=SHR
//OUT      DD DSN=*.TVBGCN.CNBCK.OUT,DISP=(OLD,KEEP,DELETE)
//SAS.SYSIN  DD DSN=XHSNER1.ROMNET1.PROCLIB(RAGTAGCN),DISP=SHR
//
```

1.10 CONC FILE QUALITY CONTROL PROCEDURES

Our simplest procedures for CONC file quality control are as follows; note that we do not include on the distribution tape the programs that accomplish these steps.

- Extract the hour-averaged ozone and tracer species for all three layers.
- Produce a "CONC" file where each cell's value is the 24-hour (or 12-hour period for the first and last days of an episode, since each episode begins and ends at 12:00 EST) maximum value; e.g., a 15-day scenario will contain values for 14 × 24 h and 2 × 12 h per cell.
- Using an NCAR[11] graphics package, we contour-plot layer 1.
- The quality control decision criteria are as follows:

  For ozone -

  - smooth, continuous data (expected since ozone is a secondary compound);
  - the existence of urban plumes;

11. National Center for Atmospheric Research, Boulder, CO.

- O_3 > background in rural areas during episodic conditions;
- boundary conditions have not significantly impacted the domain;
- high-end magnitudes are not significantly > 200 - 300 ppb
  - if the run is a base case, check against observed data
  - if the run is a control strategy, check for ozone response in the expected direction.

For tracer species[12] -

- the existence of urban plumes. Note that tracer concentrations should be virtually identical between runs with different emission control strategies but with the same meteorology.

12. We perform quality control analyses on the tracer species plots only if the ozone plots indicate that a problem may exist.

SECTION 2

CORE MODEL INPUT FILES

The Core Model expects eight files to be provided as input, four of which are produced by the processor network. These four input files are BMAT (Section 2.2), BTRK (Section 2.3), BCON (Section 2.1) and ICON (Section 2.4). The other four files are NEWICON (Section 2.5), the new initial conditions file that links one three-day scenario model run with the next; PROG (Section 2.6), the progress file that enables you to monitor the model execution; STOPCK (Section 2.8), the stop check file that lets you shut down the model during execution; and RESTRT (Section 2.7), the file that allows you to restart the model run.

We need to draw your attention to two points that apply to the following documentation. First, during the historical course of ROM development the terms "levels" and "layers" were used interchangeably and entirely synonymously. For example, one variable is referenced as NLEVS, but we use the term "layers" in this document. Second, we focus on the necessary information contained in the code; extraneous text is replaced by ellipsis marks (. . .).

We refer you to Tables E-1 and E-2 for the full list of values of the model- and region-specific parameters that are contained in DIMENS.EXT and REGION.EXT.

2.1 THE BOUNDARY CONDITIONS (BCON) FILE

The boundary conditions file (BCON) contains chemical concentrations for each of the species required by the Core Model. Concentration data are provided in each model layer for a perimeter that is one-cell deep within the modeling domain.

The boundary conditions are used by subroutine BIGGAM during calculation of the advection component of the current time step's concentration field. If the calculation is for a grid cell near the modeling domain's boundary, values from the BCON file are substituted for the previous time step's concentrations at the border cells where inflow exists.

Array dimensions are set by parameter statements contained in the INCLUDE files REGION.EXT and DIMENS.EXT as follows:

```
INTEGER*4 NROWS, NCOLS
PARAMETER ( ..., NROWS = 52, NCOLS = 64 )

INTEGER*4 NLEVS, NSPECS
PARAMETER (NLEVS = 3, NSPECS = 35, ... )
```

2.1.1 Opening the BCON File

At the start of the model scenario, BIGGAM calls BCPRCS. BCPRCS calls RDBCON, which opens the BCON file by calling OPBCON, and which optionally positions the file to the starting time for the model execution by calling POBCON. OPBCON reads the BCON file header records.

The code segments and the FORMAT statements that demonstrate the steps to open the BCON file are listed below. FLNMBC contains the internal (logical) names for BCON that point to the actual file names in the execution run stream. FLNMBC is set in the block data module BLKMOD. JUNIT is a function subprogram that returns the next available FORTRAN I/O unit number.

```
      INTEGER*4 , ..., UNITBC, ...

      CHARACTER*12 , ..., FLNMBC, ...

      SUBROUTINE OPBCON
      INTEGER*4 IOST, ..., JFILE2
      LOGICAL*4 RECFMT, RDONLY
C
      PARAMETER ( RECFMT = .FALSE., RDONLY =  .TRUE. )
C
C open BCON file
      UNITBC = JFILE2 (FLNMBC, RECFMT, RDONLY)
C
==============================================================================
      FUNCTION JFILE2 (FNAME, RECFMT, RDONLY)
      CHARACTER*12 FNAME, FORM, UNFORM, FORMAT
      INTEGER*4 IDEV, IOST, JFILE2, ..., JUNIT, ...
      LOGICAL*4 RECFMT, RDONLY
      DATA FORM / 'FORMATTED   ' /
      DATA UNFORM / 'UNFORMATTED ' /
      IDEV = JUNIT()
      IF (RECFMT) THEN
         FORMAT = FORM
           .
           .
           .
         ELSE
         FORMAT = UNFORM
           .
           .
           .
         END IF
       IF (RDONLY) THEN
         OPEN (UNIT   = IDEV,
     &         IOSTAT = IOST,
     &         FILE   = FNAME,
     &         STATUS = 'OLD',
     &         ACCESS = 'SEQUENTIAL',
     &         FORM   = FORMAT,
     &         READONLY)
           .
           .
           .
         ELSE
           .
           .
           .
         END IF
           .
           .
           .
       JFILE2 = IDEV
           .
```

```
        .
        .
      RETURN
      END
=============================================================================
```

2.1.2 BCON File Records

The structure of the BCON file conforms to the requirements for all ROM Core Model files, and therefore contains a standard file header. In addition, the file consists of a data body organized by time steps, each section of which is headed by a time step header record. Descriptions of the records containing this information are given below, and Appendix B contains a structure diagram for the BCON file.

2.1.2.1 BCON File Header Records--

The first four records contain the BCON file header that consists of the variables in the INCLUDE file HEADBC.EXT:

```
C
C     HEADBC.EXT
C
  BCON file header block
C
      CHARACTER*80 TEXTBC
      CHARACTER*8 GRDNBC
      CHARACTER*4 SPNMBC, LVNMBC
      REAL*4 SWLNBC, SWLTBC, NELNBC, NELTBC, DLONBC, DLATBC
      INTEGER*4 CDATBC, CTIMBC, SDATBC, STHRBC, TSTPBC, FRSTBC,
     &          NCOLBC, NROWBC, NLEVBC, NSPCBC, ICNTBC
C
      COMMON /CHARBC/ GRDNBC, SPNMBC(NSPECS), LVNMBC(NLEVS), TEXTBC(20)
      COMMON /HEADBC/ CDATBC, CTIMBC, SDATBC, STHRBC, TSTPBC, FRSTBC,
     &                SWLNBC, SWLTBC, NELNBC, NELTBC, DLONBC, DLATBC,
     &                NCOLBC, NROWBC, NLEVBC, NSPCBC, ICNTBC
C
```

2.1.2.1.1 Record 1-- The first record contains character strings of alphanumeric data that describe the file's contents. The data are first read (unformatted) by subroutine RDCHAR into the buffer SEG1BF. The data are then read (formatted) into the variables contained in the common blocks in the INCLUDE file HEADBC.EXT. The code segments and FORMAT statements for these steps are shown below, and the variables of record 1 are shown in Table 3.

```
      SUBROUTINE OPBCON
      CHARACTER*(8 * 13 + 4 * 5) SEG1BF
C
C read 1st segment
      CALL RDCHAR (UNITBC, SEG1BF, IOST)
C
=============================================================================
      SUBROUTINE RDCHAR (IUNIT, CHBUFF, IOST)
      INTEGER*4 IUNIT, IOST
      CHARACTER*(*) CHBUFF
      READ(IUNIT, IOSTAT = IOST) CHBUFF
```

```
      RETURN
      END
===============================================================================
C
      READ(SEG1BF, 1001, IOSTAT = IOST)
     &     CDATBC, CTIMBC, SDATBC, STHRBC, TSTPBC, FRSTBC,
     &     GRDNBC,
     &     SWLNBC, SWLTBC, NELNBC, NELTBC,
     &     DLONBC, DLATBC,
     &     NCOLBC, NROWBC, NLEVBC, NSPCBC, ICNTBC
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 5I4.4)
```

TABLE 3. BCON RECORD 1 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	CDATBC		Integer*4	File creation date as *MMDDYY*
2	CTIMBC	EST	Integer*4	File creation time as *HHMMSS*
3	SDATBC		Integer*4	Julian start date of scenario as *YYDDD*
4	STHRBC	EST	Integer*4	Start hour of scenario (00 to 23)
5	TSTPBC	s	Integer*4	Time step size for simulation
6	FRSTBC	s	Integer*4	Time to first step
7	GRDNBC		Char*8	Grid definition name
8	SWLNBC	° W	Real*4	Longitude of southwest corner of grid
9	SWLTBC	° N	Real*4	Latitude of southwest corner of grid
10	NELNBC	° W	Real*4	Longitude of northeast corner of grid
11	NELTBC	° N	Real*4	Latitude of northeast corner of grid
12	DLONBC	° W	Real*4	Grid cell longitudinal increment
13	DLATBC	° N	Real*4	Grid cell latitudinal increment
14	NCOLBC		Integer*4	Number of columns in grid
15	NROWBC		Integer*4	Number of rows in grid
			Integer*4	Number of levels
			Integer*4	Number of model species
16	NLEVBC		Integer*4	Number of levels in the simulation
17	NSPCBC		Integer*4	Number of species in the BCON file
18	ICNTBC		Integer*4	Number of text records

2.1.2.1.2 Record 2.- This record contains the list of species names for which the Core Model computes concentration outputs. The data are first read by subroutine RDCHAR into the buffer SPNMBF. The data are then copied into the SPNMBC array contained in a common block in the INCLUDE file HEADBC.EXT. These steps are listed below, and the variable of record 2 is shown in Table 4.

```
      SUBROUTINE OPBCON
      CHARACTER*(4 * NSPECS) SPNMBF
      INTEGER*4 ..., ISPC

C read the species names record
      CALL RDCHAR (UNITBC, SPNMBF, IOST)
      READ(SPNMBF, 1003, IOSTAT = IOST) (SPNMBC(ISPC), ISPC = 1, NSPCBC)
1003  FORMAT(6(10(A4))/)
```

TABLE 4. BCON RECORD 2 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$SPNMBC_k$		Char*4	Name of chemical species k †

† A list of chemical species names can be found in Table 1.

2.1.2.1.3 Record 3-- This record contains the list of Core Model layer names. The data are first read by subroutine RDCHAR into the buffer LVNMBF. The data are then copied into the LVNMBC array contained in a common block in the INCLUDE file HEADBC.EXT. These steps are listed below, and the variable of record 3 is shown in Table 5.

```
      SUBROUTINE OPBCON
      CHARACTER*(4 * NLEVS) LEVNBF
      INTEGER*4 , ..., ILEV

C read level names record
      CALL RDCHAR (UNITBC, LEVNBF, IOST)
      READ(LEVNBF, 1005, IOSTAT = IOST) (LVNMBC(ILEV), ILEV = 1, NLEVBC)
1005  FORMAT(2(10(A4))/)
```

TABLE 5. BCON RECORD 3 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$LVNMBC_L$		Char*4	Name of layer L

2.1.2.1.4 Records 4 - (4 + ICNTBC)-- These records contain descriptive text that was entered when the file was created by processor P22G. Each record consists of one 80-character string. The data are read by subroutine RDCHAR into the buffer TEXTBF, which is then copied into the TEXTBC array contained in a common block in the INCLUDE file HEADBC.EXT. These steps are listed below, and the variable of the records is shown in Table 6.

```
      SUBROUTINE OPBCON
      CHARACTER*80 TEXTBF
      INTEGER*4 , ..., ITXT

C read file text group
      DO 101 ITXT = 1, ICNTBC
      CALL RDCHAR (UNITBC, TEXTBF, IOST)
      TEXTBC(ITXT) = TEXTBF
101   CONTINUE
```

TABLE 6. BCON RECORDS 4 - (4+ICNTBC) VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$TEXTBC_n$		Char*80	Text string of *n* lines

2.1.2.2 BCON File Body Records--

After the BCON file has been opened and the file header read, BIGGAM obtains the BCON data by calling BCPRCS at the start of each model time step. BCPRCS in turn invokes RDBCON, which reads (1) the BCON file's time step headers, and (2) each boundary's data by calling subroutine RDFILE.

<u>2.1.2.2.1 Time Step Header Record</u>-- There is one time step header record for each scenario time step increment on the BCON file. Subroutine RDFILE is called to read four words of data into the RTSHBC common block. These steps are listed below, and the record's variables are shown in Table 7.

```
      REAL*4 DATBC, TIMBC, ELPBC, STPBC
      COMMON /RTSHBC/ DATBC, TIMBC, ELPBC, STPBC

      SUBROUTINE RDBCON
      INTEGER*4 IOST, WDTSH, ...
      PARAMETER ( NWDTSH = 4, ...)

C read BCON T.S.H.
      CALL RDFILE (UNITBC, NWDTSH, DATBC, IOST)
C
=============================================================================
      SUBROUTINE RDFILE(IUNIT, NWORDS, BUFFER, IOST)
      IMPLICIT NONE
      INTEGER*4 IUNIT, NWORDS, BUFFER, IOST
      DIMENSION BUFFER(NWORDS)
      READ(IUNIT, IOSTAT=IOST) BUFFER
      RETURN
      END
=============================================================================
```

TABLE 7. BCON TIME STEP HEADER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	DATBC		Real*4	Current time step Julian date as *YYDDD*
2	TIMBC	EST	Real*4	Current time step time as *HHMMSS*
3	ELPBC	s	Real*4	Elapsed time since scenario start
4	STPBC		Real*4	Current time step number on BCON file

2.1.2.2.2 Data Records-- These records are read for each time step, and contain the boundary values for species concentrations (by layer) for all modeling domain border cells. Note that the northern and southern boundaries have data referenced by columns, while the western and eastern boundaries have data referenced by rows. The code for these steps is listed below, and the data variables are shown in Table 8.

```
      REAL*4 NORTH, WEST, EAST, SOUTH
C
      COMMON /BCFILE/ WEST(NROWS, NLEVS, NSPECS),
     &                EAST(NROWS, NLEVS, NSPECS),
     &                NORTH(NCOLS, NLEVS, NSPECS),
     &                SOUTH(NCOLS, NLEVS, NSPECS)
C
      SUBROUTINE RDBCON
      INTEGER*4 , ..., NWDTSH, ROWWRD, COLWRD, ...
C define record sizes
      PARAMETER ( NWDTSH = 4,
     &            COLWRD = NLEVS * NCOLS,
     &            ROWWRD = NLEVS * NROWS )
C
C read Western boundary conditions . . . . . . . . . . . . . . species*
C
      DO 201 ISPC = 1, NSPECS
      CALL RDFILE (UNITBC, ROWWRD, WEST(1,1,ISPC), IOST)
201   CONTINUE
C
C read Eastern boundary conditions . . . . . . . . . . . . . . species*
C
      DO 301 ISPC = 1, NSPECS
      CALL RDFILE (UNITBC, ROWWRD, EAST(1,1,ISPC), IOST)
301   CONTINUE
C
C read Northern boundary conditions . . . . . . . . . . . . . . species*
C
      DO 401 ISPC = 1, NSPECS
      CALL RDFILE (UNITBC, COLWRD, NORTH(1,1,ISPC), IOST)
401   CONTINUE
C
C read Southern boundary conditions . . . . . . . . . . . . . . species*
C
      DO 501 ISPC = 1, NSPECS
      CALL RDFILE (UNITBC, COLWRD, SOUTH(1,1,ISPC), IOST)
501   CONTINUE

=========================================================================
      SUBROUTINE RDFILE(IUNIT, NWORDS, BUFFER, IOST)
      IMPLICIT NONE
      INTEGER*4 IUNIT, NWORDS, BUFFER, IOST
      DIMENSION BUFFER(NWORDS)
      READ(IUNIT, IOSTAT=IOST) BUFFER
      RETURN
      END
=========================================================================
```

TABLE 8. BCON DATA VARIABLES

Var No.	Var Name	Unit	Data Type	Description
				Chemical species concentrations for layer L and species k at the:
1	$WEST_{j,L,k}$	ppm	Real*4	western boundary for row j
2	$EAST_{j,L,k}$	ppm	Real*4	eastern boundary for row j
3	$NORTH_{i,L,k}$	ppm	Real*4	northern boundary for column i
4	$SOUTH_{i,L,k}$	ppm	Real*4	southern boundary for column i

2.2 THE B-MATRIX (BMAT) FILE

The B-matrix file contains the following data elements that are required by the Core Model for each grid cell, chemical species, and time step:

- the interlayer mass-flux coefficients (the *B-matrix coefficients*) for the parameterization of the vertical fluxes across the surfaces of the three model layers,
- the parameterizations for the emissions sources, and
- the meteorology data to make local adjustments to the Carbon Bond 4.2 (CBM4.2) chemical mechanism state constants for air temperature, atmospheric density, water vapor concentration, and altitude.

The B-matrix file requires a large amount of disk space and may have been written to several smaller subfiles on different disk packs (since each pack may not individually have had sufficient space to contain the entire file). These subfiles would then be assigned separate logical names in the job's run stream (refer to Processor P40G in Part 2 of the ROM User's Guide).

The B-matrix data are used by subroutine LILGAM for the determination of the vertical flux component and chemical reaction component in the calculation of the predicted concentration field for the next time step.

Array dimensions are set by parameter statements contained in the INCLUDE files REGION.EXT and DIMENS.EXT as follows:

```
INTEGER*4 NROWS, NCOLS
PARAMETER ( ..., NROWS = 52, NCOLS = 64 )

INTEGER*4 NLEVS, NSPECS, ..., NPOXSP
PARAMETER (NLEVS = 3, NSPECS = 35, ..., NPOXSP = 3)
```

2.2.1 Opening the BMAT File

At the start of the model scenario, LILGAM calls BMPRCS. BMPRCS calls RDBMAT, which opens the BMAT file by calling OPBMAT, and which optionally positions the file to the starting time for the model execution by calling POBMAT. RDBMAT reads the BMAT file header records by calling RDMXBM.

The code segments and the FORMAT statements that demonstrate the steps to open the BMAT file are listed below. FLNMBM contains the internal (logical) names for BMAT that point to the actual file names in the execution run stream. FLNMBM is set in the block data module BLKMOD. JUNIT is a function subprogram that returns the next available FORTRAN I/O unit number.

```
      INTEGER*4 ..., UNITBM, ...

      INTEGER*4 NUMBMF
      PARAMETER ( NUMBMF = 6)

      CHARACTER*12 ..., FLNMBM(NUMBMF), ...

      SUBROUTINE RDMXBM ( , , )
      INTEGER*4 ..., ISUB, ...
      LOGICAL*4 RECFMT, RDONLY
C
      PARAMETER ( RECFMT = .FALSE., RDONLY = .TRUE.,
     &            RECLEN = NVRBM1 * NCOLS )
C
C open 1st subfile
      UNITBM = JFILE6 (FLNMBM(ISUB), RECFMT, RDONLY, RECLEN)
C
=============================================================================
      FUNCTION JFILE6 (FNAME, RECFMT, RDONLY, RECLEN)
      CHARACTER*12 FNAME, FORM, UNFORM, FORMAT
      INTEGER*4 RECLEN, IDEV, IOST, JFILE6, JUNIT
      LOGICAL*4 RECFMT, RDONLY
      DATA FORM / 'FORMATTED   ' /
      DATA UNFORM / 'UNFORMATTED ' /
      IDEV = JUNIT()
      IF (RECFMT) THEN
         FORMAT = FORM
              .
              .
              .
         ELSE
         FORMAT = UNFORM
              .
              .
              .
         END IF
      IF (RDONLY) THEN
         OPEN (UNIT             = IDEV,
     &         IOSTAT           = IOST,
     &         FILE             = FNAME,
     &         STATUS           = 'OLD',
     &         ACCESS           = 'SEQUENTIAL',
     &         FORM             = FORMAT,
     &         CARRIAGECONTROL  = 'FORTRAN',
     &         RECORDTYPE       = 'VARIABLE',
     &         RECL             = RECLEN,
     &         READONLY)
              .
              .
              .
         ELSE
```

```
              .
              .
              .
         END IF
C
      JFILE6 = IDEV
              .
              .
              .
      RETURN
      END
==========================================================================
```

2.2.2 BMAT File Records

The structure of the B-matrix file conforms to the requirements for all ROM Core Model files, and therefore contains a standard file header. In addition, the file consists of a data body organized by time steps, each section of which is headed by a time step header record. Descriptions of the records containing this information are given below. Appendix B contains a structure diagram for the BMAT file.

2.2.2.1 BMAT File Header Records--

The first seven records contain the BMAT file header that comprise the variables in the INCLUDE file HEADBM.EXT:

```
C     HEADBM.EXT  (ROM2.1 - Carbon Bond 4.2 Chemistry)
C
C BMATRIX file header block
C
      CHARACTER*80 TEXTBM
      CHARACTER*12 MFNMBM
      CHARACTER*8 GRDNBM
      CHARACTER*4 SPNMBM, LVNMBM
      REAL*4 SWLNBM, SWLTBM, NELTBM, NELNBM, DLONBM, DLATBM
      INTEGER*4 ISUBFL, NSUBFL, FRSTSF, LSSTSF,
     &          CDATBM, CTIMBM, CDMFBM, CTMFBM, UDMFBM, UTMFBM,
     &          SDATBM, STHRBM, TSTPBM, FRSTBM, BMSPRD, BMINDX,
     &          NCOLBM, NROWBM, NLEVBM, NSPCBM, NMIFBM, ICNTBM,
     &          NMFBM, NVRBM1, NVRBM2
C
      PARAMETER (NMFBM = 4, NVRBM1 = 18 + 3 * NLEVS + 6 * NPOXSP,
     &           NVRBM2 = 6)
C
C NMFBM    number of MIF files used in generating BMATRIX file
C NVRBM1   number of part 1 BMATRIX variables,
C          specifically; B12, B13, B21, B23, B32, B33,
C                        B11S, B11SS, B31S, B31SS, QO3FAC, SSONO,
C                        TTHETA, PPS12, ZZ0, ZZ1, ZZ2, ZZ3;
C                        RRHO(NLEVS)'s, TTEMP(NLEVS)'s, WWVC(NLEVS)'s;
C                    and G1S, G1SS, G1FAC, G3S, G3SS, G3FAC, each
C                        dimensioned by NPOXSP
C NVRBM2   number of species-array BMATRIX variables, (part 2)
C          specifically; B11, B22, B31, G1, G2, G3
C
      COMMON /CHARBM/ GRDNBM, SPNMBM(NSPECS), LVNMBM(NLEVS), TEXTBM(20),
     &                MFNMBM(NMFBM)
C
      COMMON /HEADBM/ CDATBM, CTIMBM, SDATBM, STHRBM, TSTPBM, FRSTBM,
     &                SWLNBM, SWLTBM, NELNBM, NELTBM, DLONBM, DLATBM,
     &                NCOLBM, NROWBM, NLEVBM, NSPCBM, NMIFBM, ICNTBM,
     &                CDMFBM(NMFBM), CTMFBM(NMFBM), UDMFBM(NMFBM),
```

```
&               UTMFBM(NMFBM), BMINDX(NSPECS),
&               ISUBFL, NSUBFL, FRSTSF, LSSTSF
 COMMON /BMSPRD/ BMSPRD
```

2.2.2.1.1 Record 1-- The first record contains character strings of alphanumeric data that describe the file's contents. The data are first read (unformatted) into a character buffer that is then read (formatted) into the variables contained in the common blocks in the INCLUDE file HEADBM.EXT. These steps are shown below, and the variables of record 1 are shown in Table 9.

```
      INTEGER*4 ..., UNITBM, ...

      INTEGER*4 NUMBMF
      PARAMETER ( NUMBMF = 6)

      CHARACTER*12 ..., FLNMBM(NUMBMF), ...

      SUBROUTINE RDMXBM ( , , )
      INTEGER*4 ..., JFILE6, ISUB, ...
      LOGICAL*4 RECFMT, RDONLY
C
      PARAMETER ( RECFMT = .FALSE., RDONLY = .TRUE.,
     &            RECLEN = NVRBM1 * NCOLS )
C
C open 1st subfile
      UNITBM = JFILE6 (FLNMBM(ISUB), RECFMT, RDONLY, RECLEN)
C
C read subfile headers
      CALL RDHDBM (ISUB)

=========================================================================
      SUBROUTINE RDHDBM (ISUB)
      INTEGER*4 IOST, , , BFLONE, ...
      PARAMETER ( BFLONE = 13 * 8 + 6 * 4, ...)
      CHARACTER*(BFLONE) RECONE
C
C read 1st segment record
C
      READ(UNITBM, IOSTAT = IOST) RECONE
C convert character to mixed character & numeric and load HEADBM
C
      READ(RECONE, 1001, IOSTAT = IOST)
     &     CDATBM, CTIMBM, SDATBM, STHRBM, TSTPBM, FRSTBM,
     &     GRDNBM, SWLNBM, SWLTBM, NELNBM, NELTBM, DLONBM,
     &     DLATBM, NCOLBM, NROWBM, NLEVBM, NSPCBM, NMIFBM,
     &     ICNTBM
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 6I4.4)
=========================================================================
```

TABLE 9. B-MATRIX RECORD 1 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	CDATBM		Integer*4	File creation date as *MMDDYY*
2	CTIMBM	EST	Integer*4	File creation time as *HHMMSS*
3	SDATBM		Integer*4	Julian start date of scenario as *YYDDD*
4	STHRBM	EST	Integer*4	Start hour of scenario (00 to 23)
5	TSTPBM	s	Integer*4	Time step size for simulation
6	FRSTBM	s	Integer*4	Time to first step
7	GRDNBM		Char*8	Grid definition name
8	SWLNBM	° W	Real*4	Longitude of southwest corner of grid
9	SWLTBM	° N	Real*4	Latitude of southwest corner of grid
10	NELNBM	° W	Real*4	Longitude of northeast corner of grid
11	NELTBM	° N	Real*4	Latitude of northeast corner of grid
12	DLONBM	° W	Real*4	Grid cell longitudinal increment
13	DLATBM	° N	Real*4	Grid cell latitudinal increment
14	NCOLBM		Integer*4	Number of columns in grid
15	NROWBM		Integer*4	Number of rows in grid
16	NLEVBM		Integer*4	Number of levels
17	NSPCBM		Integer*4	Number of model species
18	NMIFBM		Integer*4	Number of model input files used to generate the B-matrix file
19	ICNTBM		Integer*4	Number of text records

2.2.2.1.2 Record 2-- This record contains the list of species names for which the Core Model computes concentration outputs. The data are read directly into the SPNMBM array contained in a common block in the INCLUDE file HEADBM.EXT. These steps are listed below, and the variable of record 2 is shown in Table 10.

```
      SUBROUTINE RDHDBM (ISUB)
      INTEGER*4 ..., ISPC, ...
C read species names record
C
      READ(UNITBM, IOSTAT = IOST) (SPNMBM(ISPC), ISPC = 1, NSPCBM)
```

TABLE 10. B-MATRIX RECORD 2 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$SPNMBM_k$		Char*4	Name of chemical species *k* †

† A list of chemical species names can be found in Table 1.

2.2.2.1.3 Record 3-- This record contains the expansion list (look-up table) for the 26 species names for which the BMAT file has concentration values. (Refer to Processor P23G in Part 2 of the ROM User's Guide for an explanation of the need for an expansion list.) These species names are mapped to the full Core Model list of species names. Mapping is necessary because the Core Model requires concentration values for species that have neither measured nor computed data, such as the free radicals. The expansion list tells the Core Model how to assign concentration values for the full 35 species of the chemistry mechanism from the 26 species concentrations in the BMAT file. The mapping from BMAT to the Core Model is shown in Figure 7. The READ and FORMAT statements for this record are listed below, and its variable is shown in Table 11.

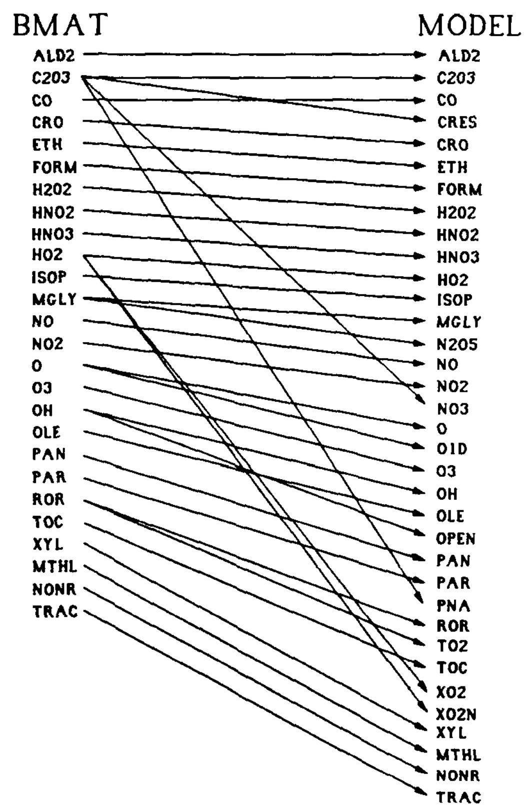


Figure 7. Chemical species mapping from BMAT to the Core Model. Species names are defined in Section 1.2.2.

```
      SUBROUTINE RDHDBM (ISUB)
      INTEGER*4 ..., BFLNDX, ...
      PARAMETER ( ..., BFLNDX = NSPECS * 4, ... )

      CHARACTER*(BFLNDX) RECNDX
C
C read the index group record
C
      READ(UNITBM, IOSTAT = IOST) RECNDX
C
```

```
C convert character to numeric and load HEADBM
C
      READ(RECNDX, 1003, IOSTAT = IOST) (BMINDX(ISPC), ISPC = 1, NSPCBM)
1003  FORMAT(<NSPCBM>I4)
```

TABLE 11. B-MATRIX RECORD 3 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$BMINDX_n$		Integer*4	Mapping array of the 26 species names whose values exist on the reduced B-matrix set, with indices for expansion to the full model set

2.2.2.1.4 Record 4-- This record contains the list of Core Model layer names. The READ and FORMAT statements for this record are listed below, and its variable is shown in Table 12.

```
      SUBROUTINE RDHDBM (ISUB)
      INTEGER*4 ..., LEV, ...
C
C read the level names record
C
      READ(UNITBM, IOSTAT = IOST) (LVNMBM(LEV), LEV = 1, NLEVBM)
```

TABLE 12. B-MATRIX RECORD 4 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$LVNMBM_L$		Char*4	Name of layer *L*

2.2.2.1.5 Records 5 - 8-- These records contain the list of logical names of the model input files (MIF) that are used to generate the B-matrix data. (The logical names can change from run to run.) The data are read (unformatted) into the RECMIF buffer and then read (formatted) into the HEADBM.EXT variables as listed below. The variables are shown in Table 13.

```
      SUBROUTINE RDHDBM (ISUB)
      INTEGER*4 ..., BFLMIF, , , IMIF, ...
C
      PARAMETER ( ..., BFLMIF = 12 + 4 * 8, ... )
      CHARACTER*(BFLMIF) RECMIF
C
C read the MIF data records
C
      DO 101 IMIF = 1, NMIFBM
      READ(UNITBM, IOSTAT = IOST) RECMIF
```

```
      READ(RECMIF, 1007, IOSTAT = IOST)
     &     MFNMBM(IMIF), CDMFBM(IMIF), CTMFBM(IMIF),
     &     UDMFBM(IMIF), UTMFBM(IMIF)
1007  FORMAT(A12, 4I8.8)
101   CONTINUE
```

TABLE 13. B-MATRIX RECORDS 5 - 8 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	$MFNMBM_n$		Char*12	File name of MIF file *n*
2	$CDMFBM_n$		Integer*4	Creation date of MIF file *n*
3	$CTMFBM_n$	EST	Integer*4	Creation time of MIF file *n*
4	$UDMFBM_n$		Integer*4	Last update date of MIF file *n*
5	$UTMFBM_n$	EST	Integer*4	Last update time of MIF file *n*

2.2.2.1.6 Records 9 - (28 maximum)-- These records contain descriptive text that was entered when the file was created by processor P40G. Each record consists of one 80-character string. The READ and FORMAT statements for these records are listed below, and their variable is shown in Table 14.

```
      SUBROUTINE RDHDBM (ISUB)
      INTEGER*4 ..., BFLTXT, , , ITXT, ...
C
      PARAMETER ( ..., BFLTXT = 80, ... )
      CHARACTER*(BFLTXT) RECTXT
C
C read header text records
C
      DO 201 ITXT = 1, ICNTBM
      READ(UNITBM, IOSTAT = IOST) RECTXT
C
      READ(RECTXT, 1009, IOSTAT = IOST) TEXTBM(ITXT)
1009  FORMAT(A80)
C
201   CONTINUE
```

TABLE 14. B-MATRIX RECORDS 9 - (9+ICNTBM-1) VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	$TEXTBM_n$		Char*80	Text string of *n* lines

2.2.2.1.7 Subfile-Order Record-- The purpose of this record is to ensure the correct ordering of input data when the Core Model reads several B-matrix subfiles. The READ and FORMAT statements for this record are listed below, and its variables are shown in Table 15.

```
      SUBROUTINE RDHDBM (ISUB)
      INTEGER*4 ..., BFLFLC, , , ISUB
C
      PARAMETER ( ..., BFLFLC = 4 * 4 )
      CHARACTER*(BFLFLC) FLCREC
C
C read subfile header record
C
      READ(UNITBM, IOSTAT = IOST) FLCREC
C
C convert character to numeric and load HEADBM
C
      READ(FLCREC, 1013, IOSTAT = IOST) ISUBFL, NSUBFL, FRSTSF, LSSTSF
1013  FORMAT(4I4)
```

TABLE 15. B-MATRIX SUBFILE-ORDER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	ISUBFL		Integer*4	Ordinal for this subfile
2	NSUBFL		Integer*4	Total number of subfiles for this B-matrix file
3	FRSTSF		Integer*4	First time step count for this subfile
4	LSSTSF		Integer*4	Last time step count for this subfile

2.2.2.2 **BMAT File Body Records--**

After the BMAT file has been opened and the file header read, LILGAM calls BMPRCS row by row to obtain the BMAT data. BMPRCS in turn invokes RDBMAT, which reads (1) the BMAT file's time step headers, and (2) each row of data by calling RDMXBM.

2.2.2.2.1 Time Step Header Record-- There is one time step header record written for each time step increment on the B-matrix file. The following code reads the time step data on the BMAT file into the time step header common block, RTSHBM, by referencing the first variable of the common block and specifying the number of the block's words to be read. These steps, and the FORMAT statements for reading this record, are listed below; the record's variables are shown in Table 16.

```
      REAL*4 DATBM, TIMBM, ELPBM, STPBM
      COMMON / RTSHBM / DATBM, TIMBM, ELPBM, STPBM

      SUBROUTINE RDBMAT (IOST)

      INTEGER*4 NWDTSH, ..., IOST
      PARAMETER ( NWDTSH = 4 ..., )
C
C read BMAT T.S.H.
      CALL RDMXBM (NWDTSH, DATBM, IOST)
C
=========================================================================
      SUBROUTINE RDMXBM (NWORDS, BUFF, IOST)
      REAL*4 BUFF
      INTEGER*4 NWORDS, IOST, ...
C
```

```
C read record
C
      READ (UNITBM, IOSTAT = IOST) BUFF
==========================================================================
```

TABLE 16. B-MATRIX TIME STEP HEADER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	DATBM		Integer*4	Current time step Julian date as *YYDDD*
2	TIMBM	EST	Integer*4	Current time step time as *HHMMSS*
3	ELPBM	s	Integer*4	Elapsed time since scenario start
4	STPBM		Integer*4	Current time step number on B-matrix file

<u>2.2.2.2.2 **Data Records**</u>-- There are two groups of data records for each B-matrix time step increment. The first group (BMAT Part 1 variables, Table 17) contains one record of variables for each model domain row. These variables consist of:

- interlayer flux parameters independent of the model chemical species (B12, B13, B21, B23, B32, B33, B11S, B11SS, B31S, B31SS,QO3FAC, SS0NO);
- ambient conditions data for the chemical rate constants adjustments (TTHETA, PPSI2, ZZ0, ZZ1, ZZ2, ZZ3, RRHO, TTEMP, WWVC); and
- volume flux emissions source parameters for the two primary oxidant species emissions, NO and NO_2 (G1S, G1SS, G1FAC, G3S, G3SS, G3FAC).

The second group (BMAT Part 2 variables, Table 18) contains a data record for each model domain row for each chemical species in the B-matrix reduced set. This set is expanded to the full model set by means of the look-up table BMINDX that is read from the file header (see Section 2.2.2.1.3). These variables consist of:

- interlayer flux parameters and volume flux emissions source parameters that are dependent on the model chemical species (B11, B22, B31, G1, G2, G3).

The following parameters from INCLUDE file HEADBM.EXT are used for dimensioning the common blocks and controlling the read operations:

NVRBM1 = 18 + 3xNLEVS + 6xNPOXSP, where NLEVS = 3 and NPOXSP = 3, the number of primary oxidant species (NO, NO_2, and O_3);

NVRBM1 is the number of Part 1 B-matrix variables, specifically: B12, B13, B21, B23, B32, B33, B11S, B11SS, B31S, B31SS, QO3FAC, SS0NO, TTHETA, PPSI2, ZZ0, ZZ1, ZZ2, ZZ3; RRHO(NLEVS)'s, TTEMP(NLEVS)'s, WWVC(NLEVS)'s; and G1S, G1SS, G1FAC, G3S, G3SS, G3FAC, each dimensioned by NPOXSP;

NVRBM2 = 6;

NVRBM2 is the number of species-array B-matrix variables (Part 2), specifically: B11, B22, B31, G1, G2, G3.

The first group (Part 1) is read into the common block BMFILE by referencing the first variable of the common block and specifying the number of the block's words to be loaded. Subsequently, the second group (Part 2) is read into the last variable in BMFILE through an iteration of all the species on the BMAT file, again by specifying the number of the variables words to be read. These steps are listed below.

```
      COMMON /BMFILE/
     &      XB12(NCOLS), XB13(NCOLS),
     &      XB21(NCOLS), XB23(NCOLS),
     &      XB32(NCOLS), XB33(NCOLS),
     &      XB11S(NCOLS),  XB11SS(NCOLS),
     &      XB31S(NCOLS),  XB31SS(NCOLS),
     &      XQO3FC(NCOLS), XSSONO(NCOLS),
     &      XG1S(NCOLS, NPOXSP), XG1SS(NCOLS, NPOXSP),
     &      XG1FAC(NCOLS, NPOXSP),
     &      XG3S(NCOLS, NPOXSP), XG3SS(NCOLS, NPOXSP),
     &      XG3FAC(NCOLS, NPOXSP),
     &      XRHO(NCOLS, NLEVS), XTEMP(NCOLS, NLEVS),
     &      XWVC(NCOLS, NLEVS),
     &      XTHETA(NCOLS), XPSI2(NCOLS),
     &      XZ0(NCOLS), XZ1(NCOLS), XZ2(NCOLS), XZ3(NCOLS),
     &      AA1(NCOLS, 6, NSPECS)
C
      SUBROUTINE RDBMAT (IOST)
      INTEGER*4 ..., NWDBM1, NWDBM2, ISPC, IOST
      PARAMETER ( ..., NWDBM1 = NVRBM1 * NCOLS, NWDBM2 = NVRBM2 * NCOLS )
C
C read BMAT part 1
      CALL RDMXBM (NWDBM1, XB12, IOST)

C read BMAT part 2 (reduced species list)
      DO 301 ISPC = 1, BMSPRD
      CALL RDMXBM (NWDBM2, AA1(1,1,ISPC), IOST)
301   CONTINUE
```

Subroutine BMPRCS moves the BMFILE data into another common block, LGBMFL. The calculations using these data are made in subroutine LILGAM, which references LGBMFL:

```
       COMMON /LGBMFL/
     &      B12(NCOLS), B13(NCOLS),
     &      B21(NCOLS), B23(NCOLS),
     &      B32(NCOLS), B33(NCOLS),
     &      B11(NCOLS, NSPECS), B22(NCOLS, NSPECS), B31(NCOLS, NSPECS),
     &      G1(NCOLS,  NSPECS), G2(NCOLS, NSPECS), G3(NCOLS, NSPECS),
     &      B11S(NCOLS),   B11SS(NCOLS),
     &      B31S(NCOLS),   B31SS(NCOLS),
     &      QO3FAC(NCOLS), SSONO(NCOLS),
     &      G1S(NCOLS, NPOXSP), G1SS(NCOLS, NPOXSP),
     &      G1FAC(NCOLS, NPOXSP),
     &      G3S(NCOLS, NPOXSP), G3SS(NCOLS, NPOXSP),
     &      G3FAC(NCOLS, NPOXSP),
     &      RHO(NCOLS, NLEVS),  TEMP(NCOLS, NLEVS), WVC(NCOLS, NLEVS),
     &      THETA(NCOLS), PSI2(NCOLS),
     &      ZLEV(NCOLS, NLEVS + 1)

       SUBROUTINE BMPRCS

C transfer BM data into model ordered arrays and
C produce LILGAM BMAT record
```

```
      DO 301 ISPC = 1, NSPCIN
      INDEX = NXSPBM(ISPC)
      DO 301 ICOL = 1, NCOLIN
      B11(ICOL, ISPC) = AA1(ICOL, 1, INDEX)
      B22(ICOL, ISPC) = AA1(ICOL, 2, INDEX)
      B31(ICOL, ISPC) = AA1(ICOL, 3, INDEX)
      G1(ICOL, ISPC) =  AA1(ICOL, 4, INDEX)
      G2(ICOL, ISPC) =  AA1(ICOL, 5, INDEX)
      G3(ICOL, ISPC) =  AA1(ICOL, 6, INDEX)
301   CONTINUE
C
C move LILGAM data from BM file organized by col & lev
      DO 401 ILEV = 1, NLEVIN
      DO 401 ICOL = 1, NCOLIN
      RHO(ICOL, ILEV)  = XRHO(ICOL, ILEV)
      TEMP(ICOL, ILEV) = XTEMP(ICOL, ILEV)
      WVC(ICOL, ILEV) = XWVC(ICOL, ILEV)
401   CONTINUE
C
C move LILGAM data from BM file organized by col & primary ox. spec.
      DO 501 ISPC = 1, NPOXSP
      DO 501 ICOL = 1, NCOLS
      G1S(ICOL, ISPC)   = XG1S(ICOL, ISPC)
      G1SS(ICOL, ISPC)  = XG1SS(ICOL, ISPC)
      G1FAC(ICOL, ISPC) = XG1FAC(ICOL, ISPC)
      G3S(ICOL, ISPC)   = XG3S(ICOL, ISPC)
      G3SS(ICOL, ISPC)  = XG3SS(ICOL, ISPC)
      G3FAC(ICOL, ISPC) = XG3FAC(ICOL, ISPC)
501   CONTINUE
C
C move remaining LILGAM data from BM file organized by col
      DO 601 ICOL = 1, NCOLIN
      B12(ICOL) = XB12(ICOL)
      B13(ICOL) = XB13(ICOL)
      B21(ICOL) = XB21(ICOL)
      B23(ICOL) = XB23(ICOL)
      B32(ICOL) = XB32(ICOL)
      B33(ICOL) = XB33(ICOL)


      B11S(ICOL)  = XB11S(ICOL)
      B11SS(ICOL) = XB11SS(ICOL)
      B31S(ICOL)  = XB31S(ICOL)
      B31SS(ICOL) = XB31SS(ICOL)
      QO3FAC(ICOL) = XQO3FC(ICOL)
      SSONO(ICOL) = XSSONO(ICOL)
      THETA(ICOL) = XTHETA(ICOL)
      PSI2(ICOL)  = XPSI2(ICOL)
      ZLEV(ICOL, 1) = XZ0(ICOL)
      ZLEV(ICOL, 2) = XZ1(ICOL)
      ZLEV(ICOL, 3) = XZ2(ICOL)
      ZLEV(ICOL, 4) = XZ3(ICOL)
601   CONTINUE
```

TABLE 17. B-MATRIX DATA VARIABLES, PART 1

Var No.	Var Name	Unit	Data Type	Description
				B-matrix coefficient in column *i* for:
1	$B12_i$	s^{-1}	Real*4	layer-1/surface-2 flux
2	$B13_i$	s^{-1}	Real*4	layer-1/surface-3 flux
3	$B21_i$	s^{-1}	Real*4	layer-2/surface-1 flux
4	$B23_i$	s^{-1}	Real*4	layer-2/surface-3 flux
5	$B32_i$	s^{-1}	Real*4	layer-3/surface-2 flux
6	$B33_i$	s^{-1}	Real*4	layer-3/surface-3 flux
				B-matrix coefficient for subgrid scale adjustment in column *i* for:
7	$B11S_i$	s^{-1}	Real*4	layer-1/surface-1 flux
8	$B11SS_i$	s^{-1}	Real*4	alternate layer-1/surface-1 flux
9	$B31S_i$	s^{-1}	Real*4	layer-3/surface-1 flux
10	$B31SS_i$	s^{-1}	Real*4	alternate layer-3/surface-1 flux
				Run time subgrid-scale adjustment parameters in column *i*:
11	$QO3FAC_i$	$m{\cdot}s^{-1}$	Real*4	ozone factor
12	$SS0NO_i$	$ppm{\cdot}m{\cdot}s^{-1}$	Real*4	NO surface emissions source strength
13	$G1S_{i,k}$	$ppm{\cdot}s^{-1}$	Real*4	emissions source term in layer 1 for primary oxidant species *k*
14	$G1SS_{i,k}$	$ppm{\cdot}s^{-1}$	Real*4	alternate emissions source term in layer 1 for primary oxidant species *k*
15	$G1FAC_{i,k}$	s^{-1}	Real*4	emissions source factor in layer 1 for primary oxidant species *k*
16	$G3S_{i,k}$	$ppm{\cdot}s^{-1}$	Real*4	emissions source term in layer 3 for primary oxidant species *k*
17	$G3SS_{i,k}$	$ppm{\cdot}s^{-1}$	Real*4	alternate emissions source term in layer 3 for primary oxidant species *k*
18	$G3FAC_{i,k}$	s^{-1}	Real*4	emissions source factor in layer 3 for primary oxidant species *k*

continued

TABLE 17 (concluded)

Var No.	Var Name	Unit	Data Type	Description
				Rate constants density correction factor in column *i* for:
19	$RHO_{i,1}$	$mol{\cdot}ppm{\cdot}m^{-3}$	Real*4	layer 1
20	$RHO_{i,2}$	$mol{\cdot}ppm{\cdot}m^{-3}$	Real*4	layer 2
21	$RHO_{i,3}$	$mol{\cdot}ppm{\cdot}m^{-3}$	Real*4	layer 3
				Absolute temperature for rate constants adjustment in column *i* for:
22	$TEMP_{i,1}$	K	Real*4	layer 1
23	$TEMP_{i,2}$	K	Real*4	layer 2
24	$TEMP_{i,3}$	K	Real*4	layer 3
				Water vapor concentration for rate constants adjustment in column *i* for:
25	$WVC_{i,1}$	ppm	Real*4	layer 1
26	$WVC_{i,2}$	ppm	Real*4	layer 2
27	$WVC_{i,3}$	ppm	Real*4	layer 3
28	$THETA_i$	deg	Real*4	Solar zenith angle for photolytic rate constants adjustment in column *i*
29	$PSI2_i$		Real*4	Cloud cover correction factor for photolytic rate constants adjustment in column *i*
				Heights above sea level in column *i* (used for rate constant adjustments):
30	$ZLEV_{i,0}$	m	Real*4	layer 0
31	$ZLEV_{i,1}$	m	Real*4	layer 1
32	$ZLEV_{i,2}$	m	Real*4	layer 2
33	$ZLEV_{i,3}$	m	Real*4	layer 3

TABLE 18. B-MATRIX DATA VARIABLES, PART 2

Var No.	Var Name	Unit	Data Type	Description
				B-matrix coefficient in column *i* for species *k* for:
34	$B11_{i,k}$	s^{-1}	Real*4	layer-1/surface-1 flux
35	$B22_{i,k}$	s^{-1}	Real*4	layer-2/surface-2 flux
36	$B31_{i,k}$	s^{-1}	Real*4	layer-3/surface-1 flux
				Emissions source term in column *i* for species *k* for:
37	$G1_{i,k}$	$ppm{\cdot}s^{-1}$	Real*4	layer 1
38	$G2_{i,k}$	$ppm{\cdot}s^{-1}$	Real*4	layer 2
39	$G3_{i,k}$	$ppm{\cdot}s^{-1}$	Real*4	layer 3

2.3 THE BACKTRACK (BTRK) FILE

The Backtrack file (BTRK) contains the gridded backtrack locations and the horizontal diffusivities data for each grid cell, model layer, and time step required by the Core Model. A backtrack location is the position in grid cell coordinates of the fluid particle that will arrive at a grid cell corner during the next 30-minute advection time step. The backtrack locations and diffusivities are used by subroutine BIGGAM in the calculation of the advection component of the current time step's concentration field.

Array dimensions are set by parameter statements contained in the INCLUDE files REGION.EXT and DIMENS.EXT as follows:

```
INTEGER*4 ..., NCOLS
PARAMETER ( ..., NCOLS = 64 )

INTEGER*4 NLEVS, ...
PARAMETER (NLEVS = 3, ...)
```

2.3.1 Opening the BTRK File

At the start of the model scenario, BIGGAM calls BTPRCS. BTPRCS calls RDBTRK, which opens the BTRK file by calling OPBTRK, and which optionally positions the file to the starting time for the model execution by calling POBTRK. RDBTRK reads the BTRK file header records by calling RDBT.

The code segments and the FORMAT statements that demonstrate the steps to open the BTRK file are listed below. FLNMBT contains the internal (logical) names for BTRK that point to the actual file names in the execution run stream. FLNMBT is set in the block data module BLKMOD. NVARBT is the number of layer-dependent BTRK variables (4). JUNIT is a function subprogram that returns the next available FORTRAN I/O unit number.

```
      INTEGER*4 ..., UNITBT, ...

      CHARACTER*12 ..., FLNMBT, ...

      SUBROUTINE RDBT (...)
      LOGICAL*4 RECFMT, RDONLY
      INTEGER*4 ..., JFILE6, RECLEN
      PARAMETER ( ...,
     &            RECLEN = NVARBT * NCOLS * NLEVS,
     &            RECFMT = .FALSE., RDONLY = .TRUE.)

C open file
      UNITBT = JFILE6 (FLNMBT, RECFMT, RDONLY, RECLEN)
C
=============================================================================
      FUNCTION JFILE6 (FNAME, RECFMT, RDONLY, RECLEN)
      CHARACTER*12 FNAME, FORM, UNFORM, FORMAT
      INTEGER*4 RECLEN, IDEV, IOST, JFILE6, JUNIT
      LOGICAL*4 RECFMT, RDONLY
      DATA FORM / 'FORMATTED   ' /
      DATA UNFORM / 'UNFORMATTED ' /
      IDEV = JUNIT()
      IF (RECFMT) THEN
         FORMAT = FORM
             .
             .
             .
         ELSE
         FORMAT = UNFORM
             .
             .
             .
         END IF
      IF (RDONLY) THEN
         OPEN (UNIT             = IDEV,
     &         IOSTAT           = IOST,
     &         FILE             = FNAME,
     &         STATUS           = 'OLD',
     &         ACCESS           = 'SEQUENTIAL',
     &         FORM             = FORMAT,
     &         CARRIAGECONTROL  = 'FORTRAN',
     &         RECORDTYPE       = 'VARIABLE',
     &         RECL             = RECLEN,
     &         READONLY)
             .
             .
             .
         ELSE
             .
             .
             .
         END IF
C
      JFILE6 = IDEV
             .
             .
             .
      RETURN
      END
=============================================================================
```

2.3.2 BTRK File Records

The structure of the Backtrack file conforms to the requirements for all ROM Core Model files, and therefore contains a standard file header. In addition, the file consists of a data body organized by time steps, each section of which is headed by a time step header record. Descriptions of the records containing this information are given below. Appendix B contains a structure diagram for the BTRK file.

2.3.2.1 BTRK File Header Records--

The first three records contain the BTRK file header that comprises of the variables in the INCLUDE file HEADBT.EXT:

```
C     HEADBT.EXT
C
C BCKTRAK file header block
C
      CHARACTER*80 TEXTBT
      CHARACTER*12 MFNMBT
      CHARACTER*8 GRDNBT
      REAL*4 SWLNBT, SWLTBT, NELTBT, NELNBT, DLONBT, DLATBT
      INTEGER*4 CDATBT, CTIMBT, CDMFBT, CTMFBT, UDMFBT, UTMFBT,
     &          SDATBT, STHRBT, TSTPBT, FRSTBT,
     &          NCOLBT, NROWBT, NMIFBT, ICNTBT,
     &          NMFBT, NVARBT
C
      PARAMETER (NMFBT = 6, NVARBT = 4)
C
C NMFBT    number of MIF files used in generating BCKTRAK file
C NVARBT   number of LEVEL-DEPENDENT BCKTRAK variables,
C          specifically; UU's, VV's, KKHU's, KKHV's
C
      COMMON /CHARBT/ GRDNBT, TEXTBT(20), MFNMBT(NMFBT)
C
      COMMON /HEADBT/ CDATBT, CTIMBT, SDATBT, STHRBT, TSTPBT, FRSTBT,
     &                SWLNBT, SWLTBT, NELNBT, NELTBT, DLONBT, DLATBT,
     &                NCOLBT, NROWBT, ICNTBT,
     &                NMIFBT,
     &                CDMFBT(NMFBT), CTMFBT(NMFBT), UDMFBT(NMFBT),
     &                UTMFBT(NMFBT)
C
```

2.3.2.1.1 Record 1-- The first record contains character strings of alphanumeric data that describe the file's contents. The data are first read (unformatted) into a character buffer that is then converted into the record variables by a formatted internal read statement. The RDBT code segments and FORMAT statements for these steps are shown below, and the variables of record 1 are shown in Table 19.

```
      SUBROUTINE RDBT (. . .)
      PARAMETER (BFLONE = 13 * 8 + 4 * 4, ...
      CHARACTER*(BFLONE) RECONE
C
C read header record
C
      READ(UNITBT, IOSTAT = IOST) RECONE
C
C convert character to mixed character & numeric
C
      READ(RECONE, 1001, IOSTAT = IOST)
     &     CDATBT, CTIMBT, SDATBT, STHRBT, TSTPBT, FRSTBT,
     &     GRDNBT, SWLNBT, SWLTBT, NELNBT, NELTBT, DLONBT,
     &     DLATBT, NCOLBT, NROWBT, NMIFBT, ICNTBT
 1001 FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4)
```

TABLE 19. BTRK RECORD 1 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	CDATBT		Integer*4	File creation date as *MMDDYY*
2	CTIMBT	EST	Integer*4	File creation time as *HHMMSS*
3	SDATBT		Integer*4	Julian start date of scenario as *YYDDD*
4	STHRBT	EST	Integer*4	Start hour of scenario (00 to 23)
5	TSTPBT	s	Integer*4	Time step size for simulation
6	FRSTBT	s	Integer*4	Time to first step
7	GRDNBT		Char*8	Grid definition name
8	SWLNBT	° W	Real*4	Longitude of southwest corner of grid
9	SWLTBT	° N	Real*4	Latitude of southwest corner of grid
10	NELNBT	° W	Real*4	Longitude of northeast corner of grid
11	NELTBT	° N	Real*4	Latitude of northeast corner of grid
12	DLONBT	° W	Real*4	Grid cell longitudinal increment
13	DLATBT	° N	Real*4	Grid cell latitudinal increment
14	NCOLBT		Integer*4	Number of columns in grid
15	NROWBT		Integer*4	Number of rows in grid
16	NMIFBT		Integer*4	Number of model input files used to generate the BTRK file
17	ICNTBT		Integer*4	Number of text records

2.3.2.1.2 Records 2 - 5-- These records contain the list of model input file logical names used to generate the BTRK file data; the list records are processed in the same way as record 1. The variables of these records are shown in Table 20.

```
      SUBROUTINE RDBT (. . .)
      PARAMETER ( ..., BFLMIF = 12 + 4 * 8, ...
      CHARACTER*(BFLMIF) RECMIF
C
C read the MIF data records
C
      DO 101 IMIF = 1, NMIFBT
      READ(UNITBT, IOSTAT = IOST) RECMIF
      READ(RECMIF, 1005, IOSTAT = IOST)
     &     MFNMBT(IMIF), CDMFBT(IMIF), CTMFBT(IMIF),
     &     UDMFBT(IMIF), UTMFBT(IMIF)
1005  FORMAT(A12, 4I8.8)
101   CONTINUE
```

TABLE 20. BTRK RECORDS 2 - 5 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	$MFNMBT_n$		Char*12	File name of MIF file *n*
2	$CDMFBT_n$		Integer*4	Creation date of MIF file *n*
3	$CTMFBT_n$	EST	Integer*4	Creation time of MIF file *n*
4	$UDMFBT_n$		Integer*4	Last update date of MIF file *n*
5	$UTMFBT_n$	EST	Integer*4	Last update time of MIF file *n*

2.3.2.1.3 Records 6 - (25 maximum)-- These records contain descriptive text that was entered when the file was created by processor P38G. Each record consists of one 80-character string. The READ and FORMAT statements in subroutine RDBT for these records are listed below, and their variable is shown in Table 21.

```
      SUBROUTINE RDBT (. . .)
      CHARACTER*80 RECTXT
C
C read header text records
C
      DO 201 ITXT = 1, ICNTBT
      READ (UNITBT, IOSTAT = IOST) RECTXT
      READ(RECTXT, 1007, IOSTAT = IOST) TEXTBT(ITXT)
1007  FORMAT(A80)
201   CONTINUE
```

TABLE 21. BTRK RECORDS 6 - (6+ICNTBT-1) VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	$TEXTBT_n$		Char*80	Text string of *n* lines

2.3.2.2 BTRK File Body Records--

After the BTRK file has been opened and the file header read, BIGGAM obtains the BTRK data by calling BTPRCS at the start of each model time step. BTPRCS in turn invokes RDBTRK, which reads (1) the BTRK file's time step headers, and (2) each row of data by calling RDBT.

2.3.2.2.1 Time Step Header Record-- There is one time step header record written for each time step increment on the BTRK file. The time step header data are contained in the common block RTSHBT. RDBT reads the data from the BTRK file by specifying the number of the block's words to be read (NWORDS) obtained from its argument list. The data are put into the output buffer DATA, which is then passed back through RDBT's argument list to

RDBTRK. The common block RTSHBT is thereby loaded since the first variable in its address space was referenced in the call to RDBT and the address space is exactly NWORDS long. The RDBTRK and RDBT code segments and the FORMAT statements that demonstrate the processing steps for this record are listed below, and its variables are shown in Table 22.

```
      SUBROUTINE RDBTRK (IOST)
      INTEGER*4 IWDLN1, ...
      PARAMETER ( IWDLN1 = 4, ...

      REAL*4 DATBT, TIMBT, ELPBT, STPBT
      COMMON / RTSHBT / DATBT, TIMBT, ELPBT, STPBT

C read BTRK T.S.H.
      CALL RDBT (IWDLN1, DATBT, IOST)
C
      SUBROUTINE RDBT (NWORDS, DATA, IOST)
      INTEGER*4 NWORDS, IOST, ...
      REAL*4 DATA
      DIMENSION DATA(NWORDS)
      READ (UNITBT, IOSTAT = IOST) DATA
```

TABLE 22. BTRK TIME STEP HEADER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	DATBT		Integer*4	Current time step Julian date as *YYDDD*
2	TIMBT	EST	Integer*4	Current time step time as *HHMMSS*
3	ELPBT	s	Integer*4	Elapsed time since scenario start
4	STPBT		Integer*4	Current time step number on BTRK file

2.3.2.2.2 Data Records-- The data records are obtained from the BTRK file in the same manner as the time step header records. RDBT reads a block of data IWDLN2 long from the file into a buffer of the same length. Since the argument list refers to the address location of the first variable in the data common block BTFILE, IWDLN2 words of file data are automatically loaded into the common block's address space. The RDBTRK and RDBT code segments and the FORMAT statements that demonstrate the processing steps for this record are listed below, and its variables are shown in Table 23.

```
C NVARBT = 4 is the number of level-dependent BCKTRAK variables,
         specifically: XRU, XRV, XRKU, XRKV.

      REAL*4 XRU, XRV, XRKU, XRKV
      COMMON /BTFILE/ XRU(NCOLS, NLEVS), XRV(NCOLS, NLEVS),
     &                XRKU(NCOLS, NLEVS), XRKV(NCOLS,NLEVS)

      SUBROUTINE RDBTRK (IOST)
      INTEGER*4 ..., IWDLN2, ...
      PARAMETER ( ..., IWDLN2 = ( NLEVS * NVARBT ) * NCOLS )

C read BTRK
      CALL RDBT (IWDLN2, XRU, IOST)
```

```
SUBROUTINE RDBT (NWORDS, DATA, IOST)
INTEGER*4 NWORDS, IOST, ...
REAL*4 DATA
DIMENSION DATA(NWORDS)
READ (UNITBT, IOSTAT = IOST) DATA
```

TABLE 23. BTRK DATA VARIABLES

Var No.	Var Name	Unit	Data Type	Description
				Backtrack locations:
1	$XRU_{i,L}$	†	Real*4	Longitudinal component, column *i*, layer *L*
2	$XRV_{i,L}$	‡	Real*4	Latitudinal component, column *i*, layer *L*
				Horizontal diffusivities:
3	$XRKU_{i,L}$	$rad^2 \cdot s^{-1}$	Real*4	Longitudinal component, column *i*, layer *L*
4	$XRKV_{i,L}$	$rad^2 \cdot s^{-1}$	Real*4	Latitudinal component, column *i*, layer *L*

† Backtrack location units are fractional column numbers.
‡ Backtrack location units are fractional row numbers.

2.4 THE INITIAL CONDITIONS (ICON) FILE

The initial conditions file (ICON) contains initial condition concentrations of all the model CBM 4.2 chemical species for every grid cell and each model layer in the model domain. These data are meant to simulate the relatively clean background conditions that would prevail prior to the onset of an elevated oxidant pollution episode. The Core Model uses these concentrations to start a multi-day simulation (an *episode*). Thus, the ICON file is used only once for each episode modeled. The ICON file is not restricted to contain data for only one time step, although in practice only one time step is read. In fact, a full concentration file (the output from the model execution) can be used as an ICON file.

The initial concentrations are used by BIGGAM only at the very first time step of the episode, during calculation of the advection component of the next time step's concentration field. For all subsequent episode time step calculations, BIGGAM uses the predicted concentrations calculated for the previous time step.

During data processing, array dimensions are set by parameter statements contained in the INCLUDE files REGION.EXT and DIMENS.EXT as follows:

```
INTEGER*4 NROWS, NCOLS
PARAMETER ( ..., NROWS = 52, NCOLS = 64 )

INTEGER*4 NLEVS, NSPECS, ..., NPOXSP
PARAMETER (NLEVS = 3, NSPECS = 35, ..., NPOXSP = 3)
```

2.4.1 Opening the ICON File

At the start of the model scenario, BIGGAM calls ICPRCS. ICPRCS calls RDICON, which opens the ICON file by calling OPICON, and which optionally positions the file to the starting time for the model execution by calling POICON. RDICON calls RDFILE to read the ICON file header records. The code segments and the FORMAT statements that demonstrate the steps to open the ICON file are listed below. Note that all ICON file records have a fixed length equal to NLEVS x NCOLS.

```
      INTEGER*4 ..., UNITIC, ...

      CHARACTER*12 ..., FLNMIC, ...

      SUBROUTINE OPICON
      INTEGER*4 IWDLTH, ..., JFILE5
      LOGICAL*4 RECFMT, RDONLY
      PARAMETER ( IWDLTH = NCOLS * NLEVS,
     &            RECFMT = .FALSE., RDONLY =  .TRUE. )
C
C open ICON file
      UNITIC = JFILE5 (FLNMIC, RECFMT, RDONLY, IWDLTH)
C
=============================================================================
      FUNCTION JFILE5 (FNAME, RECFMT, RDONLY, RECLEN)
      CHARACTER*12 FNAME, FORM, UNFORM, FORMAT
      INTEGER*4 RECLEN, IDEV, IOST, JFILE5, JUNIT, ...
      LOGICAL*4 RECFMT, RDONLY
      DATA FORM / 'FORMATTED   ' /
      DATA UNFORM / 'UNFORMATTED ' /
      IDEV = JUNIT()
      IF (RECFMT) THEN
         FORMAT = FORM
         .
         .
         .
         ELSE
         FORMAT = UNFORM
         .
         .
         .
         END IF
      IF (RDONLY) THEN
         OPEN (UNIT       = IDEV,
     &         IOSTAT     = IOST,
     &         FILE       = FNAME,
     &         STATUS     = 'OLD',
     &         ACCESS     = 'SEQUENTIAL',
     &         FORM       = FORMAT,
     &         RECORDTYPE = 'FIXED',
     &         RECL       = RECLEN,
     &         READONLY)
         .
         .
         .
         ELSE
         .
         .
         .
         END IF
         .
         .
         .
      JFILE5 = IDEV
         .
         .
         .
```

```
      RETURN
      END
==========================================================================
```

2.4.2 ICON File Records

The structure of the ICON file conforms to the requirements for all ROM Core Model files, and therefore contains a standard file header. In addition, the file consists of a data body organized by time steps, each section of which is headed by a time step header record (even though there may be only one time step on the file). Descriptions of the records containing this information are given below, and Appendix B contains a structure diagram for the ICON file.

2.4.2.1 ICON File Header Records--

The first four records contain the ICON file header that comprises the variables in the INCLUDE file HEADIC.EXT:

```
C
C     HEADIC.EXT
C
C ICON file header block
C
      CHARACTER*80 TEXTIC
      CHARACTER*8 GRDNIC
      CHARACTER*4 SPNMIC, LVNMIC
      REAL*4 SWLNIC, SWLTIC, NELNIC, NELTIC, DLONIC, DLATIC
      INTEGER*4 CDATIC, CTIMIC, SDATIC, STHRIC, TSTPIC, FRSTIC,
     &          NCOLIC, NROWIC, NLEVIC, NSPCIC, ICNTIC, IDUM
C
      COMMON /CHARIC/ GRDNIC, SPNMIC(NSPECS), LVNMIC(NLEVS), TEXTIC(20)
      COMMON /HEADIC/ CDATIC, CTIMIC, SDATIC, STHRIC, TSTPIC, FRSTIC,
     &                SWLNIC, SWLTIC, NELNIC, NELTIC, DLONIC, DLATIC,
     &                NCOLIC, NROWIC, NLEVIC, NSPCIC, ICNTIC,
     &                IDUM(8)
C
```

2.4.2.1.1 Record 1-- The first record contains character strings of alphanumeric data that describe the file's contents. The data are first read (unformatted) by subroutine RDCHAR into the buffer SEG1BF. The data are then read (formatted) into the variables contained in the common blocks in the INCLUDE file HEADIC.EXT. The code segments and FORMAT statements for these steps are shown below, and the variables of record 1 are shown in Table 24.

```
      SUBROUTINE OPICON
      CHARACTER*(8 * 21 + 4 * 5) SEG1BF
C
C read 1st segment
      CALL RDCHAR (UNITIC, SEG1BF, IOST)
C
==========================================================================
      SUBROUTINE RDCHAR (IUNIT, CHBUFF, IOST)
      INTEGER*4 IUNIT, IOST
      CHARACTER*(*) CHBUFF
      READ(IUNIT, IOSTAT = IOST) CHBUFF
```

```
      RETURN
      END
==========================================================================
C
      READ(SEG1BF, 1001, IOSTAT = IOST)
     &     CDATIC, CTIMIC, SDATIC, STHRIC, TSTPIC, FRSTIC,
     &     GRDNIC,
     &     SWLNIC, SWLTIC, NELNIC, NELTIC,
     &     DLONIC, DLATIC,
     &     NCOLIC, NROWIC, NLEVIC, NSPCIC,
     &    (IDUM(ITXT), ITXT = 1, 8),
     &     ICNTIC
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4, 8I8, I4.4)
```

TABLE 24. ICON RECORD 1 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	CDATIC		Integer*4	File creation date as *MMDDYY*
2	CTIMIC	EST	Integer*4	File creation time as *HHMMSS*
3	SDATIC		Integer*4	Julian start date of scenario as *YYDDD*
4	STHRIC	EST	Integer*4	Start hour of scenario (00 to 23)
5	TSTPIC	s	Integer*4	Time step size for simulation
6	FRSTIC	s	Integer*4	Time to first step
7	GRDNIC		Char*8	Grid definition name
8	SWLNIC	° W	Real*4	Longitude of southwest corner of grid
9	SWLTIC	° N	Real*4	Latitude of southwest corner of grid
10	NELNIC	° W	Real*4	Longitude of northeast corner of grid
11	NELTIC	° N	Real*4	Latitude of northeast corner of grid
12	DLONIC	° W	Real*4	Grid cell longitudinal increment
13	DLATIC	° N	Real*4	Grid cell latitudinal increment
14	NCOLIC		Integer*4	Number of columns in grid
15	NROWIC		Integer*4	Number of rows in grid
16	NLEVIC		Integer*4	Number of levels in the simulation
17	NSPCIC		Integer*4	Number of species in the ICON file
18	IDUM		Integer*4	Padding to fill buffer
19	ICNTIC		Integer*4	Number of text records

2.4.2.1.2 Record 2-- This record contains the list of species names for which the Core Model computes concentration outputs. The data are first read by subroutine RDCHAR into the buffer SPNMBF. The data are then copied into the SPNMIC array contained in a common block in the INCLUDE file HEADIC.EXT. These steps are listed below, and the variable of record 2 is shown in Table 25.

```
      SUBROUTINE OPICON
      CHARACTER*(4 * NSPECS) SPNMBF
      INTEGER*4 ..., ISPC, ...
```

```
C read the species names record
      CALL RDCHAR (UNITIC, SPNMBF, IOST)
      READ(SPNMBF, 1003, IOSTAT = IOST) (SPNMIC(ISPC), ISPC = 1, NSPCIC)
1003  FORMAT( <NSPECS>(A4) )
```

TABLE 25. ICON RECORD 2 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$SPNMIC_k$		Char*4	Name of chemical species k †

† A list of chemical species names can be found in Table 1.

2.4.2.1.3 Record 3-- This record contains the list of Core Model layer names. The data are first read by subroutine RDCHAR into the buffer LVNMBF. The data are then copied into the LVNMIC array contained in a common block in the INCLUDE file HEADIC.EXT. These steps are listed below, and the variable of record 3 is shown in Table 26.

```
      SUBROUTINE OPICON
      CHARACTER*(4 * NLEVS) LEVNBF
      INTEGER*4 , ..., ILEV

C read the level names record
      CALL RDCHAR (UNITIC, LEVNBF, IOST)
      READ(LEVNBF, 1005, IOSTAT = IOST) (LVNMIC(ILEV), ILEV = 1, NLEVIC)
1005  FORMAT( <NLEVS>(A4) )
```

TABLE 26. ICON RECORD 3 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$LVNMIC_L$		Char*4	Name of layer L

2.4.2.1.4 Records 4 - (4 + ICNTIC)-- These records contain descriptive text that was entered when the file was created by processor P02G. Each record consists of one 80-character string. The data are read by subroutine RDCHAR into the buffer TEXTBF, which is then copied into the TEXTIC array contained in a common block in the INCLUDE file HEADIC.EXT. These steps are listed below, and the variable of the records is shown in Table 27.

```
      SUBROUTINE OPICON
      CHARACTER*80 TEXTBF
      INTEGER*4 ..., ITXT, ...

C read file text group
      DO 101 ITXT = 1, ICNTIC
      CALL RDCHAR (UNITIC, TEXTBF, IOST)
      TEXTIC(ITXT) = TEXTBF
101   CONTINUE
```

TABLE 27. ICON RECORDS 4 - (4+ICNTIC) VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$TEXTIC_n$		Char*80	Text string of *n* lines

2.4.2.2 ICON File Body Records--

After the ICON file has been opened and the file header read, BIGGAM obtains the ICON data by calling ICPRCS before the start of the scenario time step iteration. ICPRCS, in turn, calls RDICON for each row's data.

<u>2.4.2.2.1 Time Step Header Record</u>-- There is one time step header record for each scenario time step increment on the ICON file. Usually there is only one time step on this file, but this is not an absolute restriction. Subroutine RDFILE is called to read four words of data into the RTSHIC common block by referencing the first variable of the common block and specifying the number of the block's words to be read. These steps are listed below, and the record's variables are shown in Table 28.

```
      REAL*4 DATIC, TIMIC, ELPIC, STPIC, ...
      COMMON /RTSHIC/ DATIC, TIMIC, ELPIC, STPIC, ...

      SUBROUTINE OPICON
      INTEGER*4 IOST, NWDTSH, ...
      PARAMETER (NWDTSH = 4, ...)

C read ICON T.S.H.
      CALL RDFILE (UNITIC, NWDTSH, DATIC, IOST)
C
==============================================================================
      SUBROUTINE RDFILE(IUNIT, NWORDS, BUFFER, IOST)
      IMPLICIT NONE
      INTEGER*4 IUNIT, NWORDS, BUFFER, IOST
      DIMENSION BUFFER(NWORDS)
      READ(IUNIT, IOSTAT=IOST) BUFFER
      RETURN
      END
==============================================================================
```

TABLE 28. ICON TIME STEP HEADER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	DATIC		Real*4	Current time step Julian date as *YYDDD*
2	TIMIC	EST	Real*4	Current time step time as *HHMMSS*
3	ELPIC	s	Real*4	Elapsed time since scenario start
4	STPIC		Real*4	Step number

2.4.2.2.2 **Data Records**-- These records contain the chemical species concentrations iterated over rows. The values are read into the common block ICFILE by referencing the first address of the common block for each species and specifying the number of the block's words to be read. The code for these steps is listed below, and the data variables are shown in Table 29.

```
      REAL*4 ICFILE
C
      COMMON /ICFILE/ ICFILE(NCOLS, NLEVS, NSPECS)
C

      SUBROUTINE RDICON
      INTEGER*4 IOST, ..., NWDSIC, IROW, ISPC
      PARAMETER (..., NWDSIC = NCOLS * NLEVS)

      IROW = 1
201   CONTINUE
      IF (IROW .GT. NROWIC) GO TO 301
C
C read ICON row
      DO 211 ISPC = 1, NSPECS
      CALL RDFILE (UNITIC, NWDSIC, ICFILE(1,1,ISPC), IOST)
211   CONTINUE
C
      IROW = IROW + 1
      GO TO 201
C
301   CONTINUE
```

TABLE 29. ICON DATA VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$ICFILE_{i,L,k}$	ppm	Real*4	Initial chemical species concentrations in column *i* and layer *L* for species *k*

2.5 THE NEW INITIAL CONDITIONS (NEWICON) FILE

The new initial conditions file (NEWICON) contains the predicted concentrations for the last time step of a scenario, obtained from an execution of the Core Model. NEWICON is a copy of the CONC file data for each of the 35 chemical species, each grid cell, and each of the model's 3 layers.

When the model starts the execution of the next scenario in a contiguous sequence of scenarios, the NEWICON file is used in place of the ICON file to provide initial chemical species concentrations for the first time step.

It is possible to use the CONC file generated from the preceeding scenario for initial concentration data, but since these files are large, data management becomes a serious problem for running a series of applications. The NEWICON file, containing only one time step of a CONC file, provides a solution by being much smaller than a CONC file.

During data processing, array dimensions are set by parameter statements contained in the INCLUDE files REGION.EXT and DIMENS.EXT as follows:

```
INTEGER*4 NROWS, NCOLS
PARAMETER ( ..., NROWS = 52, NCOLS = 64 )

INTEGER*4 NLEVS, NSPECS, ..., NPOXSP
PARAMETER (NLEVS = 3, NSPECS = 35, ..., NPOXSP = 3)
```

2.5.1 Opening the NEWICON File

The NEWICON file is opened at the completion of a model scenario. The main program, RUNMGR, calls subroutine NEWICS, which writes the NEWICON file header and data. The structure of NEWICON is identical to that of the ICON file. The code segments and the FORMAT statements that demonstrate the steps to open the NEWICON file are listed below. Note that all NEWICON file records have a fixed length equal to NLEVS x NCOLS.

```
      INTEGER*4 ..., UNITNI, ...

      CHARACTER*12 ..., FLNMNI, ...

      SUBROUTINE NEWICS
      INTEGER*4 IWDLTH, ..., JFILE5, ...
      LOGICAL*4 RECFMT, RDONLY
      PARAMETER ( IWDLTH = NLEVS * NCOLS,
     &            RECFMT = .FALSE., RDONLY = .FALSE.)
C
C open NEWICON file, unformatted, read/write access
      UNITNI = JFILE5 (FLNMNI, RECFMT, RDONLY, IWDLTH)

C
==============================================================================
      FUNCTION JFILE5 (FNAME, RECFMT, RDONLY, RECLEN)
      CHARACTER*12 FNAME, FORM, UNFORM, FORMAT
      INTEGER*4 RECLEN, IDEV, IOST, JFILE5, JUNIT, ...
      LOGICAL*4 RECFMT, RDONLY
      DATA FORM / 'FORMATTED   ' /
      DATA UNFORM / 'UNFORMATTED ' /
      IDEV = JUNIT()
      IF (RECFMT) THEN
         FORMAT = FORM
              .
              .
              .
         ELSE
         FORMAT = UNFORM
              .
              .
              .
         END IF
      IF (RDONLY) THEN
              .
              .
              .
```

```
      ELSE
      OPEN (UNIT        = IDEV,
     &      IOSTAT      = IOST,
     &      FILE        = FNAME,
     &      STATUS      = 'UNKNOWN',
     &      ACCESS      = 'SEQUENTIAL',
     &      FORM        = FORMAT)
      END IF
        .
        .
        .
    JFILE5 = IDEV
        .
        .
        .
    RETURN
    END
=============================================================================
```

2.5.2 NEWICON File Records

The structure of the NEWICON file conforms to the requirements for all ROM Core Model files, and therefore contains a standard file header. In addition, the file consists of a data body organized by time steps, each section of which is headed by a time step header record; note that the NEWICON file contains only one time step. Descriptions of the records containing this information are given below.

2.5.2.1 NEWICON File Header Records--

The first four records contain the NEWICON file header. The data that are written to the NEWICON file header originate from (1) header information contained in the newly generated CONC file, and (2) either the ICON or the NEWICON file used to start the scenario. In addition, (3) NEWICS calls subroutine DATTIM, which retrieves the current date and time from the the computer operating system. This time stamp becomes the creation date and time for the NEWICON file. The CONC file header data are contained in the common blocks in the HEADCN.EXT include file:

```
C
C     HEADCN.EXT
C
C CONC file header block
C
      CHARACTER*80 TEXTCN
      CHARACTER*8 GRDNCN
      CHARACTER*4 SPNMCN, LVNMCN
      REAL*4 SWLNCN, SWLTCN, NELNCN, NELTCN, DLONCN, DLATCN
      INTEGER*4 CDATCN, CTIMCN, SDATCN, STHRCN, TSTPCN, FRSTCN,
     &          NCOLCN, NROWCN, NLEVCN, NSPCCN, ICNTCN,
     &          CDBMCN, CTBMCN, CDBTCN, CTBTCN,
     &          CDBCCN, CTBCCN, CDICCN, CTICCN
C
      COMMON /CHARCN/ GRDNCN, SPNMCN(NSPECS), LVNMCN(NLEVS), TEXTCN(20)
      COMMON /HEADCN/ CDATCN, CTIMCN, SDATCN, STHRCN, TSTPCN, FRSTCN,
     &                SWLNCN, SWLTCN, NELNCN, NELTCN, DLONCN, DLATCN,
     &                NCOLCN, NROWCN, NLEVCN, NSPCCN, ICNTCN,
     &                CDBMCN, CTBMCN, CDBTCN, CTBTCN,
     &                CDBCCN, CTBCCN, CDICCN, CTICCN
C
```

2.5.2.1.1 Record 1-- The first record contains character strings of alphanumeric data that describe the file's contents. Data from the HEADCN.EXT common blocks are written to the character buffer SEG1BF, which is then written (unformatted) to NEWICON by subroutine WRCHAR. The scenario start date and time for the NEWICON file are obtained from the current (last) CONC file date and time. The variable IELPIC, which represents time elapsed on the ICON (or previous NEWICON) file from the scenario start time, is obtained from the data stored in common in the HEADIC.EXT include file (refer to Section 2.4.2.1). This variable is used to determine the time to the first step on the NEWICON file, FRSTIC. The ICON time step size, TSTPIC, is simply copied from the HEADIC.EXT common block to become the NEWICON time step size. The code segments and FORMAT statements for these steps are shown below, and the variables of record 1 are shown in Table 30.

```
      INTEGER*4 ..., UNITNI, ...

      SUBROUTINE NEWICS

C CONC header buffers
      CHARACTER*(8 * 21 + 4 * 5) SEG1BF
      INTEGER*4 ..., IOST, ...

C get creation date for new IC file
      CALL DATTIM (CDATIC, CTIMIC)
C
C get initial time values from last CONC time step
C (Note: Model cannot be started from an ICON file at half-hour)
      SDATIC = IDATCN
      STHRIC = ITIMCN / 10000
C
C get elapsed time value from ICON
      FRSTIC = IELPIC
C
C write 1st segment
      WRITE(SEG1BF, 1001, IOSTAT = IOST)
     &      CDATIC, CTIMIC, SDATIC, STHRIC, TSTPIC, FRSTIC,
     &      GRDNCN,
     &      SWLNCN, SWLTCN, NELNCN, NELTCN,
     &      DLONCN, DLATCN,
     &      NCOLCN, NROWCN, NLEVCN, NSPCCN,
     &      CDBMCN, CTBMCN,
     &      CDBTCN, CTBTCN,
     &      CDBCCN, CTBCCN,
     &      CDICCN, CTICCN,
     &      ICNTCN
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4, 8I8.8, I4.4)
C
      CALL WRCHAR (UNITNI, SEG1BF, IOST)

==============================================================================
      SUBROUTINE WRCHAR (IUNIT, CHBUF, IOST)
      IMPLICIT NONE
      INTEGER*4 IUNIT, IOST
      CHARACTER*(*) CHBUFF
      WRITE(IUNIT, IOSTAT = IOST) CHBUFF
      RETURN
      END
==============================================================================
```

TABLE 30. NEWICON RECORD 1 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	CDATIC		Integer*4	File creation date as *MMDDYY*
2	CTIMIC	EST	Integer*4	File creation time as *HHMMSS*
3	SDATIC		Integer*4	Julian start date of scenario as *YYDDD*
4	STHRIC	EST	Integer*4	Start hour of scenario (00 to 23)
5	TSTPIC	s	Integer*4	Time step size for simulation
6	FRSTIC	s	Integer*4	Time to first step
7	GRDNCN		Char*8	Grid definition name
8	SWLNCN	° W	Real*4	Longitude of southwest corner of grid
9	SWLTCN	° N	Real*4	Latitude of southwest corner of grid
10	NELNCN	° W	Real*4	Longitude of northeast corner of grid
11	NELTCN	° N	Real*4	Latitude of northeast corner of grid
12	DLONCN	° W	Real*4	Grid cell longitudinal increment
13	DLATCN	° N	Real*4	Grid cell latitudinal increment
14	NCOLCN		Integer*4	Number of columns in grid
15	NROWCN		Integer*4	Number of rows in grid
16	NLEVCN		Integer*4	Number of levels in the simulation
17	NSPCCN		Integer*4	Number of species in the NEWICON file
18	CDBMCN		Integer*4	Creation date of the B-matrix file (BMAT) from which the CONC file was generated, as *MMDDYY*
19	CTBMCN	EST	Integer*4	Creation time of the B-matrix file (BMAT) from which the CONC file was generated, as *HHMMSS*
20	CDBTCN		Integer*4	Creation date of the backtrack file (BTRK) from which the CONC file was generated, as *MMDDYY*
21	CTBTCN	EST	Integer*4	Creation time of the backtrack file (BTRK) from which the CONC file was generated, as *HHMMSS*
22	CDBCCN		Integer*4	Creation date of the boundary conditons file (BCON) from which the CONC file was generated, as *MMDDYY*
23	CTBCCN	EST	Integer*4	Creation time of the boundary conditons file (BCON) from which the CONC file was generated, as *HHMMSS*
24	CDICCN		Integer*4	Creation date of the initial conditons file (ICON) from which the CONC file was generated, as *MMDDYY*
25	CTICCN	EST	Integer*4	Creation time of the initial conditons file (ICON) from which the CONC file was generated, as *HHMMSS*
26	ICNTCN		Integer*4	Number of text records

2.5.2.1.2 Record 2-- This record contains the list of species names for which NEWICON contains concentration data. The data are first written into the buffer SPNMBF, which is then written (unformatted) to NEWICON. These steps are listed below, and the variable of record 2 is shown in Table 31.

```
      SUBROUTINE NEWICS

      CHARACTER*(4 * NSPECS) SPNMBF
      INTEGER*4 ..., ISPC, ...

      DO 101 ISPC = 1, NSPCIN
      SPNMCN(ISPC) = SPNMIN(ISPC)
101   CONTINUE

C write the species names
      WRITE(SPNMBF, 1003, IOSTAT = IOST)
     &                 (SPNMCN(ISPC), ISPC = 1, NSPCCN)
1003  FORMAT( 35(A4) )
      CALL WRCHAR (UNITNI, SPNMBF, IOST)
```

TABLE 31. NEWICON RECORD 2 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$SPNMCN_k$		Char*4	Name of chemical species k †

† A list of chemical species names can be found in Table 1.

2.5.2.1.3 Record 3-- This record contains the list of Core Model layer names. The data are first written to the buffer LVNMBF, which is then written (unformatted) to the NEWICON file. These steps are listed below, and the variable of record 3 is shown in Table 32.

```
      SUBROUTINE NEWICS

      CHARACTER*(4 * NLEVS) LEVNBF
      INTEGER*4 ..., ILEV

C write the level names
      WRITE(LEVNBF, 1005, IOSTAT = IOST)
     &                 (LVNMCN(ILEV), ILEV = 1, NLEVCN)
1005  FORMAT( 3(A4) )
      CALL WRCHAR (UNITNI, LEVNBF, IOST)
```

TABLE 32. NEWICON RECORD 3 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$LVNMCN_L$		Char*4	Name of layer L

2.5.2.1.4 Records 4 - (4 + ICNTCN)-- These records contain descriptive text that was copied to the CONC file header from the model execution runstream control parameters and now become the NEWICON descriptive text. One 80-character string is written to each record by the following statements. The variable of the records is shown in Table 33.

```
      SUBROUTINE NEWICS

      INTEGER*4 ..., ITXT, ...

C write file text group
      DO 101 ITXT = 1, ICNTCN

      CALL WRCHAR (UNITNI, TEXTCN(ITXT), IOST)
101   CONTINUE
```

TABLE 33. NEWICON RECORDS 4 - (4+ICNTCN) VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$TEXTCN_n$		Char*80	Text string of *n* lines

2.5.2.2 NEWICON File Body Records--

2.5.2.2.1 Time Step Header Record-- There is one time step header record written for the single time step increment on the NEWICON file. The time step header data is contained in the common block RTSHCN, whose first variable is used in subroutine WRFILE's argument list; note that NCOLS x NLEVS is specified as the number of words to be written. The code segments for these steps are listed below, and the record's variables are shown in Table 34.

```
      REAL*4 DATIC, TIMIC, ELPIC, STPIC, ...
      COMMON /RTSHIC/ DATIC, TIMIC, ELPIC, STPIC, ...

      SUBROUTINE NEWICS
      INTEGER*4 IWDLTH, ..., IOST, ...

      PARAMETER ( IWDLTH = NLEVS * NCOLS, ... )

C write NI T.S.H.
      CALL WRFILE (UNITNI, IWDLTH, DATIC, IOST)

=============================================================================
      SUBROUTINE WRFILE (IUNIT, NWORDS, BUFFER, IOST)
      IMPLICIT NONE
      INTEGER*4 IUNIT, NWORDS, BUFFER, IOST
      DIMENSION BUFFER(NWORDS)
      WRITE(IUNIT, IOSTAT = IOST) BUFFER
      RETURN
      END
=============================================================================
```

TABLE 34. NEWICON TIME STEP HEADER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	DATIC		Real*4	Current time step Julian date as *YYDDD*
2	TIMIC	EST	Real*4	Current time step time as *HHMMSS*
3	ELPIC	s	Real*4	Elapsed time since scenario start
4	STPIC		Real*4	Step number

<u>2.5.2.2.2 Data Records</u>-- Each data record contains the concentration data for one chemical species in one row of the domain grid for the final time step of a scenario execution. The concentration data are first copied into the ICFILE common block from the initial conditions buffer (common block BGICCN), which is written by subroutine ICPRCS at the end of each scenario step. At the end of the scenario execution, the BGICCN common block contains the last time step concentrations computed in the model. These data are written by referencing the memory address corresponding to the column 1 and level 1 indices of the array in the common block ICFILE for each species, and by passing the number of words to be written to subroutine WRFILE. The code segments for these steps are listed below and the variable for this record is shown in Table 35.

```
      REAL*4 ICFILE
      COMMON /ICFILE/ ICFILE(NCOLS, NLEVS, NSPECS)

      REAL*4 BGICCN
      INTEGER*4 TOG1, TOG2
      COMMON /BGICCN/ TOG1, TOG2, BGICCN(NCOLS, NLEVS, NSPECS, NROWS, 2)

      SUBROUTINE NEWICS
      INTEGER*4 IWDLTH, ISPC, IOST, ITXT, JFILE5,
     &          INDEX, ICOL, IROW, ILEV
      INTEGER*4 ..., IWDLN2, ..., ISPC, IOST
C
      PARAMETER ( IWDLTH = NLEVS * NCOLS, ... )

C copy body . . . . . . . . . . . . . . . . . . . . . . . . . . . . row*
C
      IROW = 1
201   CONTINUE
      IF (IROW .GT. NROWIC) GO TO 301
C
C reorder species
      DO 203 ISPC = 1, NSPCIN
      INDEX = NXSPIC(ISPC)
      DO 203 ILEV = 1, NLEVIN
      DO 203 ICOL = 1, NCOLIN
      ICFILE(ICOL, ILEV, ISPC) = BGICCN(ICOL, ILEV, INDEX, IROW, TOG2)
203   CONTINUE
C
C write NEWICON file
      DO 205 ISPC = 1, NSPECS
      CALL WRFILE(UNITNI, IWDLTH, ICFILE(1,1,ISPC),IOST)
```

```
205    CONTINUE
C
       IROW = IROW + 1
       GO TO 201
C
C copy body . . . . . . . . . . . . . . . . . . . . . . . . . . . end
C
301    CONTINUE
```

TABLE 35. NEWICON DATA VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$ICFILE_{i,L,k}$	ppm	Real*4	Initial chemical species concentrations in column i and layer L for species k

2.6 THE PROGRESS (PROG) FILE

The ROM Core Model requires a long execution time. We have found it convenient to be able to ascertain the last completed time step of a scenario during the progress of a model execution. Although this can be determined by interactively examining the currently open run log file, the progress file (PROG) is much smaller, consisting of one line, and therefore is more easily accessed.

2.6.1 Opening the PROG File

At the completion of each scenario time step, the main program, RUNMGR, opens the PROG file to write the one line time step progress data. After the data are written, RUNMGR closes the file so that there will be no conflict if a user decides to read it. In order that the computer operating system does not create new versions of the file every time it is reopened, it is necessary for the file to be opened with status 'OLD'. This, then, requires that the file exist prior to the start of a model run. The code segments below illustrate the steps to open the file.

```
      INTEGER*4 ..., UNITPR
      COMMON / LUNITS / ..., UNITPR

      CHARACTER*12 ..., FLNMPR
      COMMON / FLNAMS / ..., FLNMPR

      PROGRAM RUNMGR

C get unit numbers for . . . PROGRESS file
      UNITPR = JUNIT()

C update PROGRESS file
      OPEN (UNIT = UNITPR,
     &      STATUS = 'OLD',
     &      ACCESS = 'SEQUENTIAL',
     &      FORM = 'FORMATTED',
     &      FILE = FLNMPR)
```

2.6.2 The PROG File Record

The one line PROG file record consists of text that echoes the scenario date, scenario time, and the scenario time step number that has just been completed by the model's execution. The code segments that write this line are listed below, and the variables in the PROG record are shown in Table 36.

```
      INTEGER*4 MDDATE, MDTIME, ..., MDSTEP, ...
      COMMON /TSTEPS/ MDDATE, MDTIME, ..., MDSTEP, ...

      PROGRAM RUNMGR

      WRITE(UNITPR, 1007) MDDATE, MDTIME, MDSTEP
1007  FORMAT(3X, I6, 4X, I6.6, 4X, I4)
      WRITE(LUNOUT, 1009) MDDATE, MDTIME, MDSTEP
1009  FORMAT(/ 5X, 'Progress Completed: DATE/TIME = ', I6, ' / ', I6.6,
     &        4X, 'STEP = ', I4 )
      CLOSE (UNIT = UNITPR)
C
```

TABLE 36. PROG RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	MDDATE		Integer*4	Julian scenario date as *YYDDD*
2	MDTIME	EST	Integer*4	Scenario time as *HHMMSS*
3	MDSTEP		Integer*4	Scenario step number

2.7 THE STATE VECTOR (RESTRT) FILE

The purpose of the state vector file (RESTRT) is to allow the model run to be restarted in case a previous execution was stopped before the end of the scenario. A model run may have been stopped by either a user or the computer operating system (e.g., a queue time-out). The model can then be restarted, beginning at any time step beyond the scenario start but before the scenario step where it was stopped. This restart feature saves time and computer costs, although in our experience we have needed to use it infrequently.

The Core Model code consists of 66 subprograms in addition to the main program, RUNMGR. Of these, 14 are variable-state subroutines, which we call *processes* to distinguish them from *procedures*. Procedures execute their code from top to bottom whenever they are invoked. Processes interact with their calling programs in a critical time sequence, hence their implementation as variable-state subroutines. The processes maintain variable text pointers in their code that are set when the process is suspended and control is returned to the calling program. At the next invocation of the process, it resumes execution of the code from the point at which it was previously suspended.

RESTRT saves all necessary data to reset the variable-state subroutines to the exact state they were in when the previous run was stopped. RESTRT also contains a copy of the HEADIN.EXT common blocks, which consists of the model input data. In addition to these data, RESTRT saves the following:

- the chemistry control parameters set by subroutine INIRUN;
- the BMAT, BCON and ICON files' species look-up tables;
- species flags indicating which are the primary oxidant species;
- species flags for dealing with special species;
- diffusivities conversion factors; and
- model and file process clock data.

The first card in the user-supplied input run stream control dataset contains a text field, the requested number of steps to execute, and the starting date and time. Under normal operations, the text field is set to 'START_RUN   ' (three embedded blank spaces). To restart a model run, you must set the text field to 'RESTART_RUN ' (one embedded blank space), and also supply the starting date and time from which to continue the model execution. For example, under normal conditions for model execution, the first card will appear as follows (refer to Part 4 of the ROM User's Guide):

```
'START_RUN   ' 144 85188 120000
```

Suppose that the model execution is stopped during step 100, leaving 45 steps to process until the end of the scenario at 85191 12:00. To restart the run, you must change the first card to the following:

```
'RESTART_RUN ' 45 85190 133000
```

We recommend that you carefully count the number of time steps remaining until the end of the scenario prior to restarting the run.

In the event of a restarted run, RUNMGR calls subroutine RDSTAV to open and read the state vector file and retrieve the data necessary to reset the clocks and text pointers for all the processes (variable state subroutines). In addition, RUNMGR opens and positions the RESTRT, BCON, BTRK, and BMAT files to the time step records corresponding to the requested scenario restart time. Finally, RUNMGR calls subroutine RDCONC to get the concentration data from the CONC file for the time step prior to the restart time step. These data are then loaded into the initial conditions buffer that BIGGAM reads to calculate the advection component of the next time step's concentration field (refer to Section 2.4). With the model thus reinitialized, execution proceeds normally from the restart time step.

2.7.1 Processing that Takes Place for Normal Model Execution (how the RESTRT file gets written)

2.7.1.1 Opening the RESTRT File--

At the completion of the first model scenario time step, the main program (RUNMGR) calls subroutine WRSTAV to open the RESTRT file and record the state of the model execution on the file.

WRSTAV opens the RESTRT file with default status "formatted," allowing you to edit and change the file if you wish. The following code segments show the operations that open the file. FLNMSV is set in the block data module, BLKMOD. JUNIT is a function subprogram that returns the next FORTRAN I/O unit number not being used by the model execution.

```
      INTEGER*4 ..., UNITSV, ...
      CHARACTER*12 ..., FLNMSV, ...
      SUBROUTINE WRSTAV
      INTEGER*4 JUNIT, ..., IOST, ...
C open STATE VECTOR file, formatted, read/write access
C
      UNITSV = JUNIT()
      OPEN (UNITSV,
     &      FILE = FLNMSV,
     &      ACCESS = 'SEQUENTIAL',
     &      STATUS = 'UNKNOWN',
     &      IOSTAT = IOST)
C
```

2.7.1.2 RESTRT File Records

The structure of the RESTRT file conforms to the requirements for all ROM Core Model files, and therefore contains a standard file header. In addition, the file consists of a data body organized by time steps, each section of which is headed by a time step header record. Descriptions of the records containing this information are given below.

2.7.1.2.1 RESTRT file header records-- The first 16+ records constitute the RESTRT file header.

Just as for the CONC file header, the data that are written to the RESTRT file header originate from:

- the execution run stream's control cards;
- header information from the BMAT, BTRK, BCON, and ICON files;
- variables set by assignment statements in subroutine INIRUN; and
- arguments returned from various subroutines called by INIRUN.

These data are loaded into the common blocks in the include file HEADIN.EXT at the end of the first scenario time step of the model execution:

```
C
C     HEADIN.EXT
C
C input header information
C
C used to check file headers on BMATRIX, ICON, and BCON files
C and to create file header on RESTRT file
C
```

```
      CHARACTER*80 TEXTIN
      CHARACTER*8 GRDNIN
      CHARACTER*4 SPNMIN, LVNMIN
      REAL*4 SWLNIN, SWLTIN, NELNIN, NELTIN, DLONIN, DLATIN
      INTEGER*4 CDATIN, CTIMIN, SDATIN, STHRIN, TSTPIN, FRSTIN,
     &          NCOLIN, NROWIN, NLEVIN, NSPCIN, ICNTIN
C
      COMMON /CHARIN/ GRDNIN, SPNMIN(NSPECS), LVNMIN(NLEVS), TEXTIN(20)
      COMMON /HEADIN/ CDATIN, CTIMIN, SDATIN, STHRIN, TSTPIN, FRSTIN,
     &                SWLNIN, SWLTIN, NELNIN, NELTIN, DLONIN, DLATIN,
     &                NCOLIN, NROWIN, NLEVIN, NSPCIN, ICNTIN
C
```

The data in the HEADIN.EXT common blocks are used in the creation of the RESTRT file header. The INCLUDE file HEADSV.EXT contains the common blocks that are loaded with the RESTRT file header variables.

```
C
C     HDSTAV.EXT
C
C formatted STATE VECTOR file header block
C
      CHARACTER*80 TEXTSV
      CHARACTER*8 GRDNSV
      CHARACTER*4 SPNMSV, LVNMSV
      REAL*4 SWLNSV, SWLTSV, NELNSV, NELTSV, DLONSV, DLATSV
      INTEGER*4 CDATSV, CTIMSV, SDATSV, STHRSV, TSTPSV, FRSTSV,
     &          NCOLSV, NROWSV, NLEVSV, NSPCSV, ICNTSV
C
      COMMON /CHARSV/ GRDNSV, SPNMSV(NSPECS), LVNMSV(NLEVS), TEXTSV(20)
      COMMON /HEADSV/ CDATSV, CTIMSV, SDATSV, STHRSV, TSTPSV, FRSTSV,
     &                SWLNSV, SWLTSV, NELNSV, NELTSV, DLONSV, DLATSV,
     &                NCOLSV, NROWSV, NLEVSV, NSPCSV, ICNTSV
C
```

2.7.1.2.1.1 Record 1-- The first two records contain character strings of alphanumeric data that describe the file's contents. The data that have been stored in the common blocks in INCLUDE file HEADIN.EXT are first loaded into the common blocks in INCLUDE file HEADSV.EXT. The data are then written (formatted) to the RESTRT file. The data are written to two records (instead of one record) so that users can easily edit the file. The code segments that perform these operations are listed below, along with the WRITE and FORMAT statements. The variables of record 1 are shown in Table 37.

```
      INTEGER*4 ..., UNITSV, ...

      SUBROUTINE WRSTAV

      INTEGER*4 ..., IOST, ...

C prepare header buffer
C
      CDATSV = CDATIN
      CTIMSV = CTIMIN
      SDATSV = SDATIN
      STHRSV = STHRIN
      TSTPSV = TSTPIN
C
C first data is one time step beyond IC time step
C
      FRSTSV = FRSTIN + TSTPIN
      GRDNSV = GRDNIN
```

```
      SWLNSV = SWLNIN
      SWLTSV = SWLTIN
      NELTSV = NELTIN
      NELNSV = NELNIN
      DLONSV = DLONIN
      DLATSV = DLATIN
      NCOLSV = NCOLIN
      NROWSV = NROWIN
      NLEVSV = NLEVIN
      NSPCSV = NSPCIN
      ICNTSV = ICNTIN

C write STATE VECTOR header segment 1
C
      WRITE(UNITSV, FMT = 1001, IOSTAT = IOST)
     &      CDATSV, CTIMSV, SDATSV, STHRSV, TSTPSV,
     &      FRSTSV, GRDNSV, SWLNSV, SWLTSV, NELNSV
1001  FORMAT(1X,  2(I6, 1X), I5, 1X, I2, 1X, 2(I8, 1X),
     &       A8, 1X, 3(F8.3, 1X))
```

TABLE 37. RESTRT RECORD 1 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	CDATSV		Integer*4	File creation date as *MMDDYY*
2	CTIMSV	EST	Integer*4	File creation time as *HHMMSS*
3	SDATSV		Integer*4	Julian start date of scenario as *YYDDD*
4	STHRSV	EST	Integer*4	Start hour of scenario (00 to 23)
5	TSTPSV	s	Integer*4	Time step size for simulation
6	FRSTSV	s	Integer*4	Time to first step
7	GRDNSV		Char*8	Grid definition name
8	SWLNSV	° W	Real*4	Longitude of southwest corner of grid
9	SWLTSV	° N	Real*4	Latitude of southwest corner of grid
10	NELNSV	° W	Real*4	Longitude of northeast corner of grid

2.7.1.2.1.2 Record 2-- This record contains the remainder of the first segment data. The relevant code segments are listed below, along with the WRITE and FORMAT statements. The variables of record 2 are shown in Table 38.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., IOST, ...

C write STATE VECTOR header segment 2
C
      WRITE(UNITSV, FMT = 1003, IOSTAT = IOST)
     &      NELTSV, DLONSV, DLATSV, NCOLSV,
     &      NROWSV, NLEVSV, NSPCSV, ICNTSV
1003  FORMAT(1X, F8.3, 2(1X, F8.5), 5(1X, I4))
```

TABLE 38. RESTRT RECORD 2 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	NELTSV	° N	Real*4	Latitude of northeast corner of grid
2	DLONSV	° W	Integer*4	Grid cell longitudinal increment
3	DLATSV	° N	Integer*4	Grid cell latitudinal increment
4	NCOLSV		Integer*4	Number of columns in grid
5	NROWSV		Integer*4	Number of rows in grid
6	NLEVSV		Integer*4	Number of levels in the simulation
7	NSPCSV		Integer*4	Number of species in the RESTRT file
8	ICNTSV		Integer*4	Number of text records

2.7.1.2.1.3 Records 3 and 4-- These records contain the list of species names for which the Core Model computes concentration outputs. The data that have been stored in the HEADIN.EXT INCLUDE file common blocks are first loaded into the HEADSV.EXT common blocks. They are then written (formatted) to the RESTRT file. The data are written to two records (instead of one record) so that users can easily edit the file. The code segments that perform these operations are listed below, along with the WRITE and FORMAT statements. Note that if the number of species exceeds 30, the FORMAT statement will cause additional records to be written when the last WRITE statement is executed. The variable of records 3 and 4 is shown in Table 39.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., ISPC, IOST, ...

      DO 101 ISPC = 1, NSPCIN
      SPNMSV(ISPC) = SPNMIN(ISPC)
101   CONTINUE

C write the species names records
C
      WRITE(UNITSV, FMT = 1005, IOSTAT = IOST)
     &     (SPNMSV(ISPC), ISPC = 1, 15)
1005  FORMAT(1X, 15(A4, 1X))

      WRITE(UNITSV, FMT = 1005, IOSTAT = IOST)
     &     (SPNMSV(ISPC), ISPC = 16, NSPCSV)
```

TABLE 39. RESTRT RECORDS 3 AND 4 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$SPNMSV_k$		Char*4	Name of chemical species k †

† A list of chemical species names can be found in Table 1.

2.7.1.2.1.4 Record 5-- This record contains the list of Core Model layer names. The data that are stored in the HEADIN.EXT common blocks are first loaded into the HEADSV.EXT common blocks. They are then written (formatted) to the RESTRT file. The code segments that perform these operations are listed below, along with the WRITE and FORMAT statements. The variable of record 5 is shown in Table 40.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., IOST, ..., ILEV, ...

      DO 103 ILEV = 1, NLEVSV
      LVNMSV(ILEV) = LVNMIN(ILEV)
103   CONTINUE

C write the level names record
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1005, IOSTAT = IOST)
     &     (LVNMSV(ILEV), ILEV = 1, NLEVSV)
```

TABLE 40. RESTRT RECORD 5 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$LVNMSV_L$		Char*4	Name of layer L

2.7.1.2.1.5 Records 6 - (6 + ICNTSV)-- These records contain the descriptive text that was copied to the CONC file header from the model execution run stream control parameters. One 80-character string is written to each record. The data have been stored in the HEADIN.EXT common blocks, and are first loaded into the HEADSV.EXT common blocks. They are then written (formatted) to the RESTRT file using the code segments listed below. The variable of the records is shown in Table 41.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., IOST, ITXT, ...

      DO 105 ITXT = 1, ICNTSV
      TEXTSV(ITXT) = TEXTIN(ITXT)
105   CONTINUE

C write header text records
C
      DO 109 ITXT = 1, ICNTSV
      WRITE(UNITSV, FMT = 1007, IOSTAT = IOST) TEXTSV(ITXT)
1007  FORMAT(1X, A80)

109   CONTINUE
```

TABLE 41. RESTRT RECORDS 6 - (6+ICNTSV) VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$TEXTSV_n$		Char*80	Text string of *n* lines

2.7.1.2.1.6 HEADIN record 1-- The next two records contain copies of the HEADIN.EXT common blocks. The data are written to two records (instead of one record) so that users can easily edit the file. The WRITE and FORMAT statements for this record are listed below, and its variables are shown in Table 42.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., IOST, ...

C write HEADIN header record segment 1
C
      WRITE(UNITSV, FMT = 1001, IOSTAT = IOST)
     &      CDATIN, CTIMIN, SDATIN, STHRIN, TSTPIN, FRSTIN,
     &      GRDNIN, SWLNIN, SWLTIN, NELNIN
```

TABLE 42. RESTRT HEADIN RECORD 1 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	CDATIN		Integer*4	File creation date as *MMDDYY*
2	CTIMIN	EST	Integer*4	File creation time as *HHMMSS*
3	SDATIN		Integer*4	Julian start date of scenario as *YYDDD*
4	STHRIN	EST	Integer*4	Start hour of scenario (00 to 23)
5	TSTPIN	s	Integer*4	Time step size for simulation
6	FRSTIN	s	Integer*4	Time to first step
7	GRDNIN		Char*8	Grid definition name
8	SWLNIN	° W	Real*4	Longitude of southwest corner of grid
9	SWLTIN	° N	Real*4	Latitude of southwest corner of grid
10	NELNIN	° W	Real*4	Longitude of northeast corner of grid

2.7.1.2.1.7 HEADIN record 2-- This record is the continuation of the HEADIN.EXT common block's data. The WRITE and FORMAT statements for this record are listed below, and its variables are shown in Table 43.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., IOST, ...

C write HEADIN header record segment 2
C
      WRITE(UNITSV, FMT = 1003, IOSTAT = IOST)
     &      NELTIN, DLONIN, DLATIN, NCOLIN, NROWIN,
     &      NLEVIN, NSPCIN, ICNTIN
```

TABLE 43. RESTRT HEADIN RECORD 2 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	NELTIN	° N	Real*4	Latitude of northeast corner of grid
2	DLONIN	° W	Integer*4	Grid cell longitudinal increment
3	DLATIN	° N	Integer*4	Grid cell latitudinal increment
4	NCOLIN		Integer*4	Number of columns in grid
5	NROWIN		Integer*4	Number of rows in grid
6	NLEVIN		Integer*4	Number of levels in the simulation
7	NSPCIN		Integer*4	Number of species in the RESTRT file
8	ICNTIN		Integer*4	Number of text records

2.7.1.2.1.8 Chemistry control records-- The next two records contain the chemistry control data used by subroutine LILGAM. These data are loaded into the CHEMIN common block by subroutine INIRUN at the start of a model run. They are saved in the RESTRT file to preserve the same data values in the event that a restart run is required. The data are written to two records (instead of one record) so that users can easily edit the file. The variables of these records are shown in Table 44 and Table 45, and the code segments that write them to the RESTRT file are listed below.

```
      INTEGER*4 NCOUT, ISPEC
      REAL*4 ATS, GTS, UFRAX, BFRAX, FACTOR, DIVP, DIVQ,
     &       ULIM, BLIM, FNOLIM
C
      COMMON /CHEMIN/ ATS, GTS, UFRAX, BFRAX, FACTOR, DIVP, DIVQ,
     &                NCOUT, ISPEC(NSPECS), ULIM, BLIM, FNOLIM

      SUBROUTINE WRSTAV

      INTEGER*4 ...., IOST, ...

C Write CHEMIN header record
C
      WRITE(UNITSV, FMT = 1013, IOSTAT = IOST)
     &      ATS, GTS, UFRAX, BFRAX, FACTOR,
     &      DIVP, DIVQ, NCOUT
1013  FORMAT(1X, 2(F8.2, 1X), 5(F8.5, 1X), I5)

      WRITE(UNITSV, FMT = 1015, IOSTAT = IOST)
     &      (ISPEC(ISPC), ISPC = 1, NCOUT), ULIM, BLIM, FNOLIM
1015  FORMAT(1X, <NCOUT>(I4.3, 1X), 3(E10.3, 1X) )
```

TABLE 44. RESTRT CHEMISTRY CONTROL VARIABLES 1

Var No.	Var Name	Unit	Data Type	Description
1	ATS	s	Real*4	Advection time step
2	GTS	s	Real*4	G-tilde time step
3	UFRAX		Real*4	Upper FRAX limit
4	BFRAX		Real*4	Lower FRAX limit
5	FACTOR		Real*4	Threshold factor to include species in chemistry time step determination
6	DIVP		Real*4	Chemistry decay term predictor - corrector combination factor
7	DIVQ		Real*4	Chemistry source term predictor - corrector combination factor
8	NCOUT		Integer*4	Number of primary oxidant species

TABLE 45. RESTRT CHEMISTRY CONTROL VARIABLES 2

Var No.	Var Name	Unit	Data Type	Description
1	$ISPEC_k$		Integer*4	Index for species *k* used in chemistry time step determination
2	ULIM	s	Real*4	Chemistry time step upper limit
3	BLIM	s	Real*4	Chemistry time step lower limit
4	FNOLIM	s^{-1}	Real*4	Chemistry solution accuracy control parameter

2.7.1.2.1.9 BMAT species index records-- These records contain the expansion list (look-up table) of species names for which the B-matrix file has values. (Refer to P23G, Part 2 of the ROM User's Guide, for an explanation of the expansion list.) These values are mapped to the full Core Model list of species names (see Section 2.2.2.1.3). The WRITE and FORMAT statements for these records are listed below; note that if the number of species exceeds 30, the FORMAT statement will cause additional records to be written when the last WRITE statement is executed. The records' variable is shown in Table 46.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., ISPC, IOST, ...

C write species ordering header record
C
```

```
      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPBM(ISPC), ISPC = 1, 15)
1019  FORMAT (1X, 15(I3, 1X))

      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPBM(ISPC), ISPC = 16, NSPECS)
```

TABLE 46. BMAT SPECIES INDEX VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$NXSPBM_k$		Integer*4	BMAT file index for species *k* †

† *A list of chemical species names can be found in Table 1.*

<u>2.7.1.2.1.10 BCON species index records</u>-- These records contain the expansion list (look-up table) of species names for which the BCON file has values. These values are mapped to the full Core Model list of species (see Section 2.2.2.1.3). For ROM2.1, the BCON file contains the same species list as the Core Model, therefore the mapping table is one-to-one. The WRITE and FORMAT statements for these records are listed below; note that if the number of species exceeds 30, the FORMAT statement will cause additional records to be written when the last WRITE statement is executed. The records' variable is shown in Table 47.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ...., ISPC, IOST, ...

      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPBC(ISPC), ISPC = 1, 15)

      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPBC(ISPC), ISPC = 16, NSPECS)
1019  FORMAT (1X, 15(I3, 1X))
```

TABLE 47. BCON SPECIES INDEX VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$NXSPBC_k$		Integer*4	BCON file index for species *k* †

† *A list of chemical species names can be found in Table 1.*

<u>2.7.1.2.1.11 ICON species index records</u>-- These records contain the expansion list (look-up table) of species names for which the ICON file has values. These values are mapped to the full Core Model list of species. For ROM2.1, the ICON file contains the same species list as the Core Model, therefore the mapping table is one-to-one. The WRITE and FORMAT statements for these records are listed below; note that if the

number of species exceeds 30, the FORMAT statement will cause additional records to be written when the last WRITE statement is executed. The records' variable is shown in Table 48.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., ISPC, IOST, ...

      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPIC(ISPC), ISPC = 1, 15)

      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPIC(ISPC), ISPC = 16, NSPECS)
1019  FORMAT (1X, 15(I3, 1X))
```

TABLE 48. ICON SPECIES INDEX VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$NXSPIC_k$		Integer*4	ICON file index for species k †

† A list of chemical species names can be found in Table 1.

2.7.1.2.1.12 Primary oxidant species flag records-- These records contain LOGICAL variables corresponding to the list of model species indices. The variable has the LOGICAL value of .TRUE. if the species index corresponds to a primary oxidant species, i.e., NO, NO_2, or O_3; otherwise the variable has the logical value of .FALSE.. The WRITE and FORMAT statements for these records are listed below; note that if the number of species exceeds 30, the FORMAT statement will cause additional records to be written when the last WRITE statement is executed. The records' variable is shown in Table 49.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., ISPC, IOST, ...

C write species.control for LILGAM
C
      WRITE(UNITSV, FMT = 1025, IOSTAT = IOST)
     &     (INBIG3(ISPC), ISPC = 1, 15)
1025  FORMAT(1X, 15(L4, 1X))

      WRITE(UNITSV, FMT = 1025, IOSTAT = IOST)
     &     (INBIG3(ISPC), ISPC = 16, NSPECS)
```

TABLE 49. PRIMARY OXIDANT SPECIES FLAG RECORDS VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$INBIG3_k$		Logical*4	Primary oxidant flag for species k

2.7.1.2.1.13 Special species record-- This record contains the indices for NO, NO_2, O_3, PAR, TRAC, and NONR from the model species list. These variables are used in the Core Model for special processing of these species. The WRITE and FORMAT statements for this record are listed below, and the record's variable is shown in Table 50.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., IOST, ...

      WRITE(UNITSV, FMT = 1027, IOSTAT = IOST)
     &      NOHIT, NO2HIT, O3HIT, PARHIT, TRCHIT, NONHIT
1027  FORMAT(1X, 6(I3, 1X))
```

TABLE 50. SPECIAL SPECIES RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	NOHIT		Integer*4	Species list index for NO
2	NO2HIT		Integer*4	Species list index for NO_2
3	O3HIT		Integer*4	Species list index for O_3
4	PARHIT		Integer*4	Species list index for PAR
5	TRCHIT		Integer*4	Species list index for TRAC
6	NONHIT		Integer*4	Species list index for NONR

2.7.1.2.1.14 Diffusivities conversion factor record-- These variables are used in BIGGAM for the treatment of horizontal diffusion in the advection scheme used in the model. They must be saved to the RESTRT file because, in the event of a restart, the part of the BIGGAM code where they are set is bypassed. The WRITE and FORMAT statements for this record are listed below, and the record's variable is shown in Table 51.

```
      SUBROUTINE WRSTAV

      INTEGER*4 ..., IOST, ...

C Write diffusivities conversion factor for BIGGAM
C
      WRITE(UNITSV, FMT = 1029, IOSTAT = IOST) RDLNT2, RDLTT2
1029  FORMAT(1X, E13.6, 2X, E13.6)
```

TABLE 51. DIFFUSIVITIES CONVERSION FACTOR RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	RDLNT2	$s \cdot rad^{-2}$	Real*4	Longitudinal horizontal diffusivities conversion factor
2	RDLTT2	$s \cdot rad^{-2}$	Real*4	Latitudinal horizontal diffusivities conversion factor

2.7.1.2.2 RESTRT file body records-- At the completion of each model scenario time step the main program (RUNMGR) calls subroutine WRSTAV to record the state of the model execution on the RESTRT file. RESTRT thus contains a state vector for each completed time step.

2.7.1.2.2.1 Time step header record-- There is one time step header record for each time step increment on the RESTRT file. The code references the time step data in the time step header common block, TSHDSV, and writes it (formatted) to the RESTRT file. The common block is loaded by the main program, RUNMGR, prior to calling subroutine WRSTAV. These steps are listed below, and the record's variables are shown in Table 52.

```
      INTEGER*4 IDATSV, ITIMSV, IELPSV, ISTPSV
C
      COMMON /TSHDSV/ IDATSV, ITIMSV, IELPSV, ISTPSV

      SUBROUTINE WRSTAV
      INTEGER*4 ..., IOST, ...

      WRITE(UNITSV, FMT = 1031, IOSTAT = IOST)
     &      IDATSV, ITIMSV, IELPSV, ISTPSV
1031  FORMAT(1X, I5, 1X, I6, 1X, I8, 1X, I4)
C
```

TABLE 52. RESTRT TIME STEP HEADER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	IDATSV		Integer*4	Current scenario Julian date as *YYDDD*
2	ITIMSV	EST	Integer*4	Current scenario time as *HHMMSS*
3	IELPSV	s	Integer*4	Elapsed time since scenario start
4	ISTPSV		Integer*4	Step number

2.7.1.2.2.2 Text pointers record-- This record contains the current time step value for the 14 variable-state subroutine text pointers. The WRITE and FORMAT statements are listed below, and the record's variables are shown in Table 53.

```
C write text pointers
C
      WRSVPT = 2
      WRITE(UNITSV, FMT = 1033, IOSTAT = IOST)
     &      BIGMPT, LILGPT,
     &      BCPSPT, BMPSPT, BTPSPT, CNPSPT, ICPSPT,
     &      RDBCPT, RDBMPT, RDBTPT, RDCNPT, RDICPT,
     &      WRCNPT, WRSVPT
1033  FORMAT(1X, 14(I3, 1X))
```

TABLE 53. TEXT POINTER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	BIGMPT		Integer*4	Text pointer in subroutine BIGGAM
2	LILGPT		Integer*4	Text pointer in subroutine LILGAM
3	BCPSPT		Integer*4	Text pointer in subroutine BCPRCS
4	BMPSPT		Integer*4	Text pointer in subroutine BMPRCS
5	BTPSPT		Integer*4	Text pointer in subroutine BTPRCS
6	CNPSPT		Integer*4	Text pointer in subroutine CNPRCS
7	ICPSPT		Integer*4	Text pointer in subroutine ICPRCS
8	RDBCPT		Integer*4	Text pointer in subroutine RDBCON
9	RDBMPT		Integer*4	Text pointer in subroutine RDBMAT
10	RDBTPT		Integer*4	Text pointer in subroutine RDBTRK
11	RDCNPT		Integer*4	Text pointer in subroutine RDCONC
12	RDICPT		Integer*4	Text pointer in subroutine RDICON
13	WRCNPT		Integer*4	Text pointer in subroutine WRCONC
14	WRSVPT		Integer*4	Text pointer in subroutine WRSTAV

2.7.1.2.2.3 Row counters record-- In addition to text pointers, some of the subroutines also maintain variable indices that point to the row value of the grid currently being executed. These pointers are also maintained in the state vector file. The intent of the original design concept was to enable the model to be restarted not only at the start of a time step but also within a time step on a row boundary. However, in ROM2.1 this feature was not fully implemented. Therefore, ROM2.1 can be restarted only at time step boundaries. The WRITE and FORMAT statements are listed below, and the record's variables are shown in Table 54.

```
C write row counters
C
      WRITE(UNITSV, FMT = 1035, IOSTAT = IOST)
     &      BMPSRW, RDBMRW, RDBTRW, RDCNRW
1035  FORMAT(1X, 4(I3, 1X))
```

TABLE 54. ROW COUNTERS RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	BMPSRW		Integer*4	Row counter in subroutine BMPRCS
2	RDBMRW		Integer*4	Row counter in subroutine RDBMAT
3	RDBTRW		Integer*4	Row counter in subroutine RDBTRK
4	RDCNRW		Integer*4	Row counter in subroutine RDCONC

2.7.1.2.2.4 Model and file time step header records-- These two records save the process clock steps of the principal processes (variable-state subroutines) that (1) maintain the model scenario time (BIGGAM), and (2) manage the ROM's data files (BCPRCS, BMPRCS, BTPRCS, CNPRCS, and ICPRCS. These data ensure that the files are kept in step with one another. The WRITE and FORMAT statements are listed below, and the records' variables are shown in Table 55.

```
C write scenario time
C
      WRITE(UNITSV, FMT = 1037, IOSTAT = IOST)
     &       MDDATE, MDTIME, MDELAP, MDSTEP,
     &       BCDATE, BCTIME, BCELAP, BCSTEP,
     &       BMDATE, BMTIME, BMELAP, BMSTEP
1037  FORMAT(1X, 3(I5, 1X, I6, 1X, I8, I4, 1X))
C
      WRITE(UNITSV, FMT = 1037, IOSTAT = IOST)
     &       BTDATE, BTTIME, BTELAP, BTSTEP,
     &       CNDATE, CNTIME, CNELAP, CNSTEP,
     &       ICDATE, ICTIME, ICELAP, ICSTEP
```

TABLE 55. MODEL AND FILE TIME STEP HEADER RECORDS VARIABLES

Var No.	Var Name	Unit	Data Type	Description [a]
1	MDDATE		Integer*4	Current scenario date for MODEL
2	MDTIME	EST	Integer*4	Current scenario time for MODEL
3	MDELAP	s	Integer*4	Time elapsed since start for MODEL
4	MDSTEP		Integer*4	Current scenario step for MODEL
5	BCDATE		Integer*4	Current scenario date for BCON file
6	BCTIME	EST	Integer*4	Current scenario time for BCON file
7	BCELAP	s	Integer*4	Time elapsed since start for BCON file
8	BCSTEP		Integer*4	Current scenario step for BCON file
9	BMDATE		Integer*4	Current scenario date for BMAT file
10	BMTIME	EST	Integer*4	Current scenario time for BMAT file
11	BMELAP	s	Integer*4	Time elapsed since start for BMAT file
12	BMSTEP		Integer*4	Current scenario step for BMAT file
13	BTDATE		Integer*4	Current scenario date for BTRK file
14	BTTIME	EST	Integer*4	Current scenario time for BTRK file
15	BTELAP	s	Integer*4	Time elapsed since start for BTRK file
16	BTSTEP		Integer*4	Current scenario step for BTRK file
17	CNDATE		Integer*4	Current scenario date for CONC file
18	CNTIME	EST	Integer*4	Current scenario time for CONC file
19	CNELAP	s	Integer*4	Time elapsed since start for CONC file
20	CNSTEP		Integer*4	Current scenario step for CONC file
21	ICDATE		Integer*4	Current scenario date for ICON file
22	ICTIME	EST	Integer*4	Current scenario time for ICON file
23	ICELAP	s	Integer*4	Time elapsed since start for ICON file
24	ICSTEP		Integer*4	Current scenario step for ICON file

[a] All dates are Julian, i.e., *YYDDD*; all times are as *HHMMSS*.

2.7.2 Processing That Takes Place for Restarting Model Execution (how the RESTRT file gets read)

The Core Model main program, RUNMGR, calls subroutine RDSTAV to open and position the RESTRT file. RDSTAV does this by calling subroutines OPSTAV and POSTAV. OPSTAV opens the file and extracts the header data in the same manner as described above for subroutine WRSTAV. POSTAV positions the file to the requested scenario date and time. RDSTAV then reads the RESTRT time step header, the text pointers records (Section 2.7.1.2.2.2), the row counters record (Section 2.7.1.2.2.3), and the model and file time step header records (Section 2.7.1.2.2.4), used to reset the clocks maintained by the principal processes. The code segments for these steps are shown below.

```
      PROGRAM RUNMGR

      INTEGER*4 ..., IDATE, ITIME, ...

C open and read state vector file
      CALL RDSTAV (IDATE, ITIME)
```

```
C
================================================================================
      SUBROUTINE RDSTAV (IDATE, ITIME)

C open STATE VECTOR file
      CALL OPSTAV
C
C position STATE VECTOR file
      CALL POSTAV (IDATE, ITIME)
C
C read STATE VECTOR T.S.H.
      READ(UNITSV, FMT = 1001, IOSTAT = IOST)
     &     IDATSV, ITIMSV, IELPSV, ISTPSV
1001  FORMAT(1X, I5, 1X, I6, 1X, I8, 1X, I4)

C read text pointers
      READ(UNITSV, FMT = 1003, IOSTAT = IOST)
     &     BIGMPT, LILGPT,
     &     BCPSPT, BMPSPT, BTPSPT, CNPSPT, ICPSPT,
     &     RDBCPT, RDBMPT, RDBTPT, RDCNPT, RDICPT,
     &     WRCNPT, WRSVPT
1003  FORMAT(1X, 14(I3, 1X))

C read row counters
      READ(UNITSV, FMT = 1005, IOSTAT = IOST)
     &     BMPSRW, RDBMRW, RDBTRW, RDCNRW
1005  FORMAT(1X, 4(I3, 1X))

C read scenario time
      READ(UNITSV, FMT = 1007, IOSTAT = IOST)
     &     MDDATE, MDTIME, MDELAP, MDSTEP,
     &     BCDATE, BCTIME, BCELAP, BCSTEP,
     &     BMDATE, BMTIME, BMELAP, BMSTEP
1007  FORMAT(1X, 3(I5, 1X, I6, 1X, I8, I4, 1X))

      READ(UNITSV, FMT = 1007, IOSTAT = IOST)
     &     BTDATE, BTTIME, BTELAP, BTSTEP,
     &     CNDATE, CNTIME, CNELAP, CNSTEP,
     &     ICDATE, ICTIME, ICELAP, ICSTEP
================================================================================
```

Once the RESTRT file has been opened, positioned, and read, RUNMGR opens and positions the BCON, BTRK, and BMAT files to the time step records corresponding to the requested scenario restart time. Finally, RUNMGR calls subroutine RDCONC to obtain the concentration data for each grid row from the CONC file for the time step prior to the restart time step. RDCONC calls subroutine RDFILE to read NWDSCN words of data into the CNFILE common block. RUNMGR copies these data into the initial conditions buffer (common block BGICCN) that BIGGAM reads to calculate the advection component of the next time step's concentration field (refer to Section 2.5.2.2.2). With the model thus reinitialized, execution proceeds normally from the restart time step. The essential steps are shown below.

```
      REAL*4 BGICCN
      INTEGER*4 TOG1, TOG2
      COMMON /BGICCN/ TOG1, TOG2, BGICCN(NCOLS, NLEVS, NSPECS, NROWS, 2)

      REAL*4 CNFILE
      COMMON /CNFILE/ CNFILE(NCOLS, NLEVS, NSPECS)

      PROGRAM RUNMGR

      INTEGER*4 ..., IDATE, ITIME, ..., IROW, ISPC, LEV, ICOL, ...

C open CONC file and check file header with STATE VECTOR file header
        CALL OPCONC
             .
             .
             .
C open BCON file and check file header
        CALL OPBCON
             .
             .
             .
C open BTRK file and check file header
        CALL OPBTRK
             .
             .
             .
C open BMAT file and check file header
        CALL OPBMAT
             .
             .
             .
C position CONC file
        CALL POCONC (IDATE, ITIME)
             .
             .
             .
C position BCON file
        CALL POBCON (IDATE, ITIME)
             .
             .
             .
C position BTRK file
        CALL POBTRK (IDATE, ITIME)
             .
             .
             .
C position BMAT file
        CALL POMXBM (IDATE, ITIME)
             .
             .
             .
C copy CONC rows to ICCN file
        DO 101 IROW = 1, NROWIN
        CALL RDCONC (IOST)
        DO 101 ISPC = 1, NSPECS
        DO 101 LEV = 1, NLEVS
        DO 101 ICOL = 1, NCOLS
        BGICCN(ICOL,LEV,ISPC,IROW,TOG2) = CNFILE(ICOL,LEV,ISPC)
101     CONTINUE
C
==============================================================================
      COMMON /CNFILE/ CNFILE(NCOLS, NLEVS, NSPECS)

      SUBROUTINE RDCONC (IOST)

      INTEGER*4 IOST, ..., NWDSCN, ISPC

C define record sizes
      PARAMETER (..., NWDSCN = NCOLS * NLEVS)
```

```
      DO 211 ISPC = 1, NSPECS
      CALL RDFILE (UNITCN, NWDSCN, CNFILE(1,1,ISPC), IOST)
         .
         .
         .
211   CONTINUE
C
===============================================================================
      SUBROUTINE RDFILE(IUNIT, NWORDS, BUFFER, IOST)
      INTEGER*4 IUNIT, NWORDS, BUFFER, IOST
      DIMENSION BUFFER(NWORDS)

      READ(IUNIT, IOSTAT=IOST) BUFFER
===============================================================================
```

2.8 THE STOP CHECK (STOPCK) FILE

The stop check (STOPCK) file allows you to shut down a model execution before the run-stream-designated scenario end time. The STOPCK file can be edited during the course of a model run and its one line of text altered. If you alter the text to "STOP", then, at the end of the current time step, the model writes a NEWICON file and exits.

2.8.1 Opening the STOPCK File

At the completion of each scenario time step, the main program, RUNMGR, opens the STOPCK file to read the one line of text. If the text is "STOP", (no leading or trailing blanks), then RUNMGR shuts the model down. Otherwise, RUNMGR closes the file, and model execution continues until the end of the scenario prescribed by the run control parameters. The file is closed so that that there will be no conflict if a user decides to change the file. There is some risk involved if a user does edit the file to change it. If the model execution is coincidentally at the point where it must open the STOPCK file to check its text, and a user already has the file open to edit it, the execution will be aborted by the system. We consider this risk minimal because the fraction of time users may have the file open is small compared with the elapsing clock time required for one model step. In order that the computer operating system does not create new versions of the file every time it is reopened, it is necessary for the file to be opened with status 'OLD'. This, then, requires that the file exist prior to the start of a model run. The code segments below illustrate the steps to open the file.

```
      INTEGER*4 ..., UNITST, ...
      COMMON / LUNITS / ..., UNITST, ...

      CHARACTER*12 ..., FLNMPR
      COMMON / FLNAMS / ..., FLNMPR

      PROGRAM RUNMGR

C get unit numbers for STOP FLAG file . . .
      UNITST = JUNIT()
```

```
C check STOP flag
      OPEN (UNIT = UNITST,
     &       STATUS = 'OLD',
     &       ACCESS = 'SEQUENTIAL',
     &       FORM = 'FORMATTED',
     &       FILE = FLNMST)
```

2.8.2 The STOPCK File Record

The one line STOPCK file record consists of text that controls the continuation of the model run. The data variable is loaded into the STOPFG common block from BLOCK DATA at the start of a model run. The variable remains unchanged unless a user edits the STOPCK file and changes it. Anything other than "STOP" permits the model run to continue. The relevant code segments are listed below, and the variable in the STOPCK record is shown in Table 56.

```
      CHARACTER*4 STOPFG
      COMMON /STOPFG/ STOPFG

      BLOCK DATA BLKMOD
      DATA STOPFG / 'GO  ' /

      PROGRAM RUNMGR

      READ(UNITST, 1011) STOPFG
1011  FORMAT(A4)
      CLOSE (UNIT = UNITST)
      IF (STOPFG .EQ. 'STOP') THEN
         WRITE(LUNOUT, 1013) STOPFG
1013     FORMAT(/ 5X, '> > >STOP FLAG IS SET TO: ',A4)
         GO TO 401
         END IF


401   CONTINUE
C
C write NEW ICON file
      CALL NEWICS
C
      WRITE(LUNOUT, 1015)
1015  FORMAT(// 10X, '.... Model Run Completion from RUNMGR ....'//)
C
      STOP '            '
```

TABLE 56. STOPCK RECORD VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	STOPFG		Char *4	Flag that can terminate model execution

This page is intentionally left blank.

SECTION 3

THE CORE MODEL CONCENTRATION (CONC) OUTPUT FILE

The concentration file (CONC) contains the concentrations predicted by the ROM for each of the 35 chemical species, each grid cell, each of the model's three layers, and each time step recorded for one execution of the Core Model. The chemical species list consists of 35 species, 33 of which are the condensed species required by the Carbon Bond 4.2 chemical mechanism; the other two species are a tracer species used for quality assurance monitoring and a nonreactive hydrocarbon species.

When the model starts the execution for each scenario, the scenario initial conditions (either the ICON or the NEWICON file) are used to start the computation for the subsequent steps in the scenario. These initial concentration data are copied to the CONC file as the "zeroth" step. The first computed step, i.e., one time step beyond the initial conditions, is counted as step one. Each succeeding step is incremented by one; a CONC file for a typical three-day scenario will therefore have 145 steps, starting at day 1, hour 12 and ending at day 4, hour 12.

During processing of this file, array dimensions are set by parameter statements contained in the INCLUDE files REGION.EXT and DIMENS.EXT as follows:

```
INTEGER*4 NROWS, NCOLS
PARAMETER ( ..., NROWS = 52, NCOLS = 64 )

INTEGER*4 NLEVS, NSPECS, ..., NPOXSP
PARAMETER (NLEVS = 3, NSPECS = 35, ..., NPOXSP = 3)
```

3.1 OPENING THE CONC FILE

The CONC file is opened at the start of the model scenario. Once the model has started, subroutine BIGGAM calls LILGAM, which calls CNPRCS, which then calls WRCONC; WRCONC finally calls OPWRCN to open the file (using the JFILES function subprogram) and write its header records. Note that all CONC file records have a fixed record length equal to NLEVS x NCOLS.

FLNMCN contains the internal (logical) names for CONC that point to the actual file names in the execution run stream. FLNMCN is set in the block data module BLKMOD. JUNIT is a function subprogram that returns the next FORTRAN I/O unit number not being used by the model execution (refer to Section 1.7). The code segments and the FORMAT statements that describe the processing steps to open the CONC file are listed below.

```
      INTEGER*4 ..., UNITCN, ...

      CHARACTER*12 ..., FLNMCN, ...

      SUBROUTINE OPWRCN
      INTEGER*4 IWDLTH, ..., JFILE5
      LOGICAL*4 RECFMT, RDONLY
      PARAMETER ( IWDLTH = NLEVS * NCOLS,
     &            RECFMT = .FALSE., RDONLY = .FALSE.)
C
C open concentration file, unformatted, read/write access
      UNITCN = JFILE5 (FLNMCN, RECFMT, RDONLY, IWDLTH)
C
=========================================================================
      FUNCTION JFILE5 (FNAME, RECFMT, RDONLY, RECLEN)
      CHARACTER*12 FNAME, FORM, UNFORM, FORMAT
      INTEGER*4 RECLEN, IDEV, IOST, JFILE5, JUNIT, ...
      LOGICAL*4 RECFMT, RDONLY
      DATA FORM / 'FORMATTED   ' /
      DATA UNFORM / 'UNFORMATTED ' /
      IDEV = JUNIT()
      IF (RECFMT) THEN
         FORMAT = FORM
              .
              .
              .
         ELSE
         FORMAT = UNFORM
              .
              .
              .
         END IF
      IF (RDONLY) THEN
              .
              .
              .
         ELSE
         OPEN (UNIT       = IDEV,
     &         IOSTAT     = IOST,
     &         FILE       = FNAME,
     &         STATUS     = 'UNKNOWN',
     &         ACCESS     = 'SEQUENTIAL',
     &         FORM       = FORMAT)
         END IF
          .
          .
          .
      JFILE5 = IDEV
          .
          .
          .
      RETURN
      END
=========================================================================
```

3.2 CONC FILE RECORDS

The structure of the CONC file conforms to the requirements for all ROM Core Model files, and therefore contains a standard file header. In addition, the file consists of a data body organized by time steps, each section of which is headed by a time step header record. Descriptions of the records containing this information are given below, while Appendix B contains a structure diagram for the CONC file.

3.2.1 CONC File Header Records

The data that are written to the CONC file header originate from:

- the execution runstream's control cards,
- header information from the BMAT, BTRK, BCON, and ICON files,
- variables set by assignment statements in subroutine INIRUN, and
- arguments returned from various subroutines called by INIRUN.

These data are loaded into the common blocks in the INCLUDE file HEADIN.EXT at the beginning of the model execution:

```
C
C     HEADIN.EXT
C
C input header information
C
C used to check file headers on BMATRIX, ICON, and BCON files
C and to create file header on CONC file
C
      CHARACTER*80 TEXTIN
      CHARACTER*8 GRDNIN
      CHARACTER*4 SPNMIN, LVNMIN
      REAL*4 SWLNIN, SWLTIN, NELNIN, NELTIN, DLONIN, DLATIN
      INTEGER*4 CDATIN, CTIMIN, SDATIN, STHRIN, TSTPIN, FRSTIN,
     &          NCOLIN, NROWIN, NLEVIN, NSPCIN, ICNTIN
C
      COMMON /CHARIN/ GRDNIN, SPNMIN(NSPECS), LVNMIN(NLEVS), TEXTIN(20)
      COMMON /HEADIN/ CDATIN, CTIMIN, SDATIN, STHRIN, TSTPIN, FRSTIN,
     &                SWLNIN, SWLTIN, NELNIN, NELTIN, DLONIN, DLATIN,
     &                NCOLIN, NROWIN, NLEVIN, NSPCIN, ICNTIN
C
```

The data in the above common blocks are used in the creation of the CONC file header. The INCLUDE file HEADCN.EXT contains the common blocks that are loaded with the CONC file header variables:

```
C
C     HEADCN.EXT
C
C CONC file header block
C
      CHARACTER*80 TEXTCN
      CHARACTER*8 GRDNCN
      CHARACTER*4 SPNMCN, LVNMCN
      REAL*4 SWLNCN, SWLTCN, NELNCN, NELTCN, DLONCN, DLATCN
      INTEGER*4 CDATCN, CTIMCN, SDATCN, STHRCN, TSTPCN, FRSTCN,
     &          NCOLCN, NROWCN, NLEVCN, NSPCCN, ICNTCN,
     &          CDBMCN, CTBMCN, CDBTCN, CTBTCN,
     &          CDBCCN, CTBCCN, CDICCN, CTICCN
C
      COMMON /CHARCN/ GRDNCN, SPNMCN(NSPECS), LVNMCN(NLEVS), TEXTCN(20)
      COMMON /HEADCN/ CDATCN, CTIMCN, SDATCN, STHRCN, TSTPCN, FRSTCN,
     &                SWLNCN, SWLTCN, NELNCN, NELTCN, DLONCN, DLATCN,
     &                NCOLCN, NROWCN, NLEVCN, NSPCCN, ICNTCN,
     &                CDBMCN, CTBMCN, CDBTCN, CTBTCN,
     &                CDBCCN, CTBCCN, CDICCN, CTICCN
C
```

The first four records comprise the CONC file header.

3.2.1.1 Record 1--

The first record contains character strings of alphanumeric data that describe the file's contents. The data are first loaded into the HEADCN.EXT common blocks. They are then written to the character buffer SEG1BF, which is then written (unformatted) to the CONC file by subroutine WRCHAR. The code segments for these steps are shown below, and the variables of record 1 are shown in Table 57.

```
      INTEGER*4 ..., UNITCN, ...

      SUBROUTINE OPWRCN

C CONC header buffers
      CHARACTER*(8 * 21 + 4 * 5) SEG1BF
      INTEGER*4 ..., IOST, ...

C load CONC header
      CDATCN = CDATIN
      CTIMCN = CTIMIN
      SDATCN = SDATIN
      STHRCN = STHRIN
      TSTPCN = TSTPIN
      FRSTCN = FRSTIN
      GRDNCN = GRDNIN
      SWLNCN = SWLNIN
      SWLTCN = SWLTIN
      NELTCN = NELTIN
      NELNCN = NELNIN
      DLONCN = DLONIN
      DLATCN = DLATIN
      NCOLCN = NCOLIN
      NROWCN = NROWIN
      NLEVCN = NLEVIN
      NSPCCN = NSPCIN
      ICNTCN = ICNTIN + 1
C
C set values for creation dates/times of BM, BT, BC and IC files
      CDBMCN = CDATBM
      CTBMCN = CTIMBM
      CDBTCN = CDATBT
      CTBTCN = CTIMBT
      CDBCCN = CDATBC
      CTBCCN = CTIMBC
      CDICCN = CDATIC
      CTICCN = CTIMIC
C
C write 1st segment
      WRITE(SEG1BF, 1001, IOSTAT = IOST)
     &      CDATCN, CTIMCN, SDATCN, STHRCN, TSTPCN, FRSTCN,
     &      GRDNCN,
     &      SWLNCN, SWLTCN, NELNCN, NELTCN,
     &      DLONCN, DLATCN,
     &      NCOLCN, NROWCN, NLEVCN, NSPCCN,
     &      CDBMCN, CTBMCN,
     &      CDBTCN, CTBTCN,
     &      CDBCCN, CTBCCN,
     &      CDICCN, CTICCN,
     &      ICNTCN
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4, 8I8.8, I4.4)
C
      CALL WRCHAR (UNITCN, SEG1BF, IOST)
```

```
================================================================================
      SUBROUTINE WRCHAR (IUNIT, CHBUF, IOST)
      IMPLICIT NONE
      INTEGER*4 IUNIT, IOST
      CHARACTER*(*) CHBUFF
      WRITE(IUNIT, IOSTAT = IOST) CHBUFF
      RETURN
      END
================================================================================
```

TABLE 57. CONC RECORD 1 VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	CDATCN		Integer*4	File creation date as *MMDDYY*
2	CTIMCN	EST	Integer*4	File creation time as *HHMMSS*
3	SDATCN		Integer*4	Julian start date of scenario as *YYDDD*
4	STHRCN	EST	Integer*4	Start hour of scenario (00 to 23)
5	TSTPCN	s	Integer*4	Time step size for simulation
6	FRSTCN	s	Integer*4	Time to first step
7	GRDNCN		Char*8	Grid definition name
8	SWLNCN	° W	Real*4	Longitude of southwest corner of grid
9	SWLTCN	° N	Real*4	Latitude of southwest corner of grid
10	NELNCN	° W	Real*4	Longitude of northeast corner of grid
11	NELTCN	° N	Real*4	Latitude of northeast corner of grid
12	DLONCN	° W	Real*4	Grid cell longitudinal increment
13	DLATCN	° N	Real*4	Grid cell latitudinal increment
14	NCOLCN		Integer*4	Number of columns in grid
15	NROWCN		Integer*4	Number of rows in grid
16	NLEVCN		Integer*4	Number of levels in the simulation
17	NSPCCN		Integer*4	Number of species in the CONC file
18	CDBMCN		Integer*4	Creation date of B-matrix file (BMAT) from which the CONC file was generated (*MMDDYY*)
19	CTBMCN	EST	Integer*4	Creation time of B-matrix file (BMAT) from which the CONC file was generated (*HHMMSS*)
20	CDBTCN		Integer*4	Creation date of backtrack file (BTRK) from which the CONC file was generated (*MMDDYY*)
21	CTBTCN	EST	Integer*4	Creation time of backtrack file (BTRK) from which the CONC file was generated (*HHMMSS*)
22	CDBCCN		Integer*4	Creation date of boundary conditions (BCON) file from which the CONC file was generated (*MMDDYY*)
23	CTBCCN	EST	Integer*4	Creation time of boundary conditions (BCON) file from which the CONC file was generated (*HHMMSS*)
24	CDICCN		Integer*4	Creation date of initial conditions (CONC) file from which the CONC file was generated (*MMDDYY*)
25	CTICCN	EST	Integer*4	Creation time of initial conditions (CONC) file from which the CONC file was generated (*HHMMSS*)
26	ICNTCN		Integer*4	Number of text records

3.2.1.2 Record 2--

This record contains the list of species names for which the Core Model computes concentration outputs. The data are written to the character buffer SPNMBF, which is then written (unformatted) to the CONC file. These steps are listed below, and the variable of record 2 is shown in Table 58.

```
      SUBROUTINE OPWRCN

      CHARACTER*(4 * NSPECS) SPNMBF
      INTEGER*4 ..., ISPC, ..., IOST, ...

      DO 101 ISPC = 1, NSPCIN
      SPNMCN(ISPC) = SPNMIN(ISPC)
101   CONTINUE

C write the species names
      WRITE(SPNMBF, 1003, IOSTAT = IOST)
     &                   (SPNMCN(ISPC), ISPC = 1, NSPCCN)
1003  FORMAT( <NSPECS>(A4) )
      CALL WRCHAR (UNITCN, SPNMBF, IOST)
```

TABLE 58. CONC RECORD 2 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$SPNMCN_k$		Char*4	Name of chemical species k †

† A list of chemical species names can be found in Table 1.

3.2.1.3 Record 3--

This record contains the list of Core Model layer names. The data are written to the character buffer LEVNBF, which is then written (unformatted) to the CONC file. The steps are listed below, and the variable of record 3 is shown in Table 59.

```
      SUBROUTINE OPWRCN

      CHARACTER*(4 * NLEVS) LEVNBF
      INTEGER*4 ..., ILEV, ..., IOST, ...

      LVNMCN(1) = '   1'
      LVNMCN(2) = '   2'
      LVNMCN(3) = '   3'

C write the level names
      WRITE(LEVNBF, 1005, IOSTAT = IOST)
     &                   (LVNMCN(ILEV), ILEV = 1, NLEVCN)
1005  FORMAT( <NLEVS>(A4) )
      CALL WRCHAR (UNITCN, LEVNBF, IOST)
```

TABLE 59. CONC RECORD 3 VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$LVNMCN_L$		Char*4	Name of layer *L*

3.2.1.4 Records 4 - (4 + ICNTCN)--

These records contain descriptive text. The first text record is created by subroutine OPWRCN and contains the model version name. The subsequent records consist of descriptive text to be copied to the CONC file header. The text data were optionally entered as part of the model execution run stream (e.g., see Section 4.3 for an IBM run stream). One 80-character string is written to each record by the following statements. The variable of the records is shown in Table 60.

```
      SUBROUTINE OPWRCN

      INTEGER*4 ..., ITXT, ...
C
C copy input text records to CONC file
      TEXTCN(1) = 'ROM2.1  '
      CALL WRCHAR (UNITCN, TEXTCN(1), IOST)
C
      DO 103 ITXT = 2, ICNTCN
      TEXTCN(ITXT) = TEXTIN(ITXT - 1)
      CALL WRCHAR (UNITCN, TEXTCN(ITXT), IOST)
103   CONTINUE
```

TABLE 60. CONC RECORDS 4 - (4+ICNTCN) VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$TEXTCN_n$		Char*80	Text string of *n* lines

3.2.2 CONC File Body Records

3.2.2.1 Time Step Header Record--

At the start of each model time step, subroutine LILGAM calls CNPRCS, which in turn calls WRCONC to write the CONC file time step header. There is one time step header record written for each time step increment on the CONC file. The time step header data are contained in the common block RTSHCN. The following code segments illustrate how the data are written to the

CONC file. The first variable of the common block RTSHCN is used in subroutine WRFILE's argument list, and NCOLS x NLEVS is specified as the number of words to be written. The record's variables are shown in Table 61.

```
      COMMON / RTSHCN / DATCN, TIMCN, ELPCN, STPCN

      SUBROUTINE WRCONC
      INTEGER*4 IWLTSH, ..., IOST

C define record lengths
      PARAMETER ( IWLTSH = NCOLS * NLEVS, ... )

      CALL WRFILE (UNITCN, IWLTSH, DATCN, IOST)

==============================================================================
      SUBROUTINE WRFILE (IUNIT, NWORDS, BUFFER, IOST)
      IMPLICIT NONE
      INTEGER*4 IUNIT, NWORDS, BUFFER, IOST
      DIMENSION BUFFER(NWORDS)
      WRITE(IUNIT, IOSTAT = IOST) BUFFER
      RETURN
      END
==============================================================================
```

TABLE 61. CONC TIME STEP HEADER RECORD VARIABLES

Var No.	Var Name	Unit	Data Type	Description
1	DATCN		Real*4	Current time step date as *YYDDD*
2	TIMCN		Real*4	Current time step time as *HHMMSS*
3	ELPCN	s	Real*4	Elapsed time since scenario start
4	STPCN		Real*4	Step number on the CONC file

3.2.2.2 Data Records--

Each data record contains the concentration data for one chemical species in one row of the domain grid. Subroutine LILGAM calls CNPRCS after it completes the calculations for the vertical flux and the chemical reaction components of the concentration for all layers and all species in one row of the grid. CNPRCS calls WRCONC, which calls WRFILE to write that row of data to the CONC file for the current time step.

For each time step increment in the CONC file there are NSPECS records for each of the NROWS domain rows. These data are written by referencing the memory address corresponding to the column 1 and level 1 indices of the array in the common block CNFILE for each species, and passing the number of words to be written to WRFILE. The code segments for these steps are listed below, and the variable for these records is shown in Table 62.

```
      COMMON /CNFILE/ CNFILE(NCOLS, NLEVS, NSPECS)

      SUBROUTINE WRCONC
      INTEGER*4 ..., IWDLN2, ..., ISPC, IOST
C
C define record lengths
      PARAMETER ( ..., IWDLN2 = NCOLS * NLEVS )

C write CONC row
      DO 301 ISPC = 1, NSPECS
      CALL WRFILE (UNITCN, IWDLN2, CNFILE(1,1,ISPC), IOST)
301   CONTINUE
```

TABLE 62. CONC DATA VARIABLE

Var No.	Var Name	Unit	Data Type	Description
1	$CNFILE_{i,L,k}$	ppm	Real*4	Chemical species concentrations in column i and layer L for species k

This page is intentionally left blank.

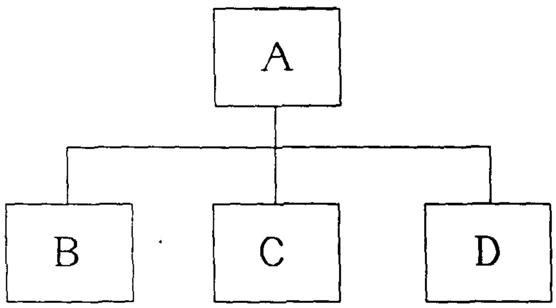
APPENDIX A

JACKSON STRUCTURED PROGRAMMING (JSP)

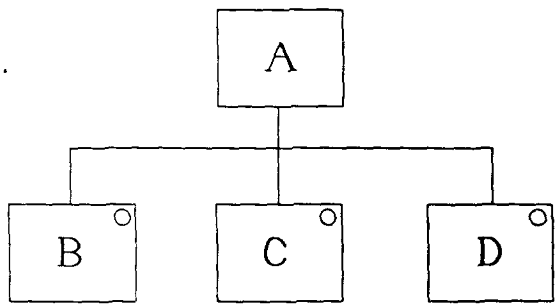
We wrote much of the ROM Core and Processor codes using the JSP methodology because it allows us to (1) eliminate the need for intermediate temporary files, (2) it eliminates the need for our knowing exactly when, and in what order, to call subroutines and functions (this scheduling is accomplished automatically), and (3) for future model upgrades, its modular format permits us to easily add or delete processors as we find necessary.

A useful JSP transformation is to write a stand-alone program, then *invert* it. We show examples of program inversion in Section A.2. Section A.1 gives you some preliminary information about JSP. For a full discussion of JSP, see Jackson (1975).

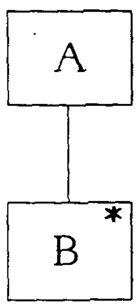
A.1 AN INTRODUCTION TO JSP FLOW DIAGRAMS



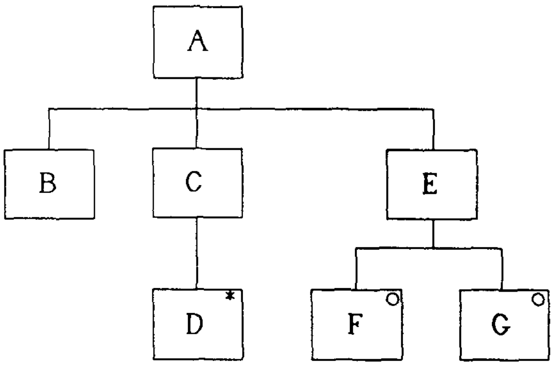
A is a *sequence* of **B** followed by **C** followed by **D**



A is a *selection* of either **B** or **C** or **D**

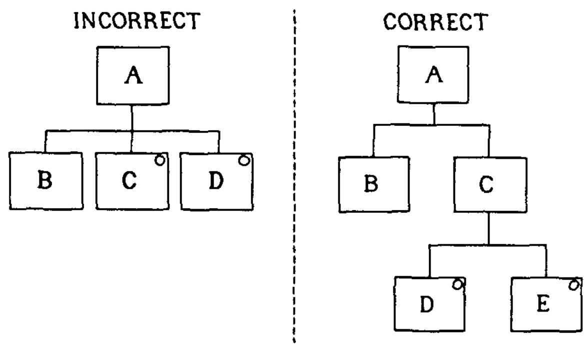


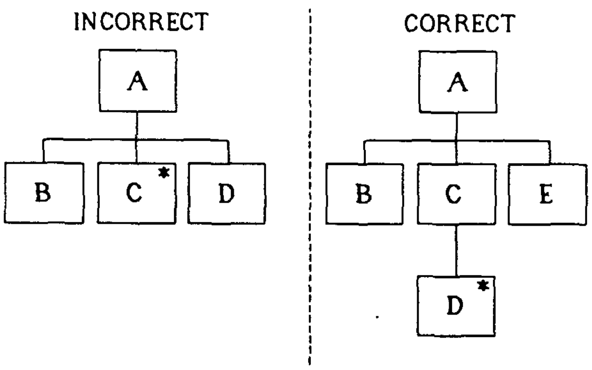
A is an *iteration* of **B** zero or more times



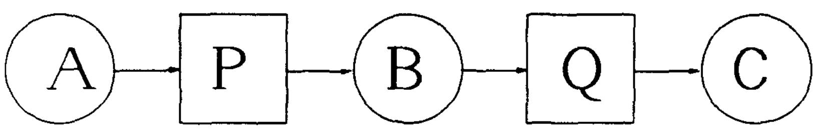
A is a sequence of B followed by C (which is an iteration of D), followed by E (which is a selection of either F or G

The next two diagrams show you two common diagramming errors and their corrected versions.



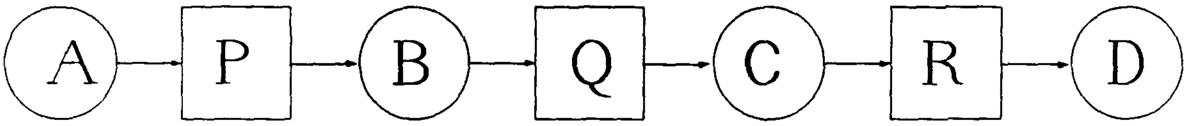


We next show a simple specification flow diagram.

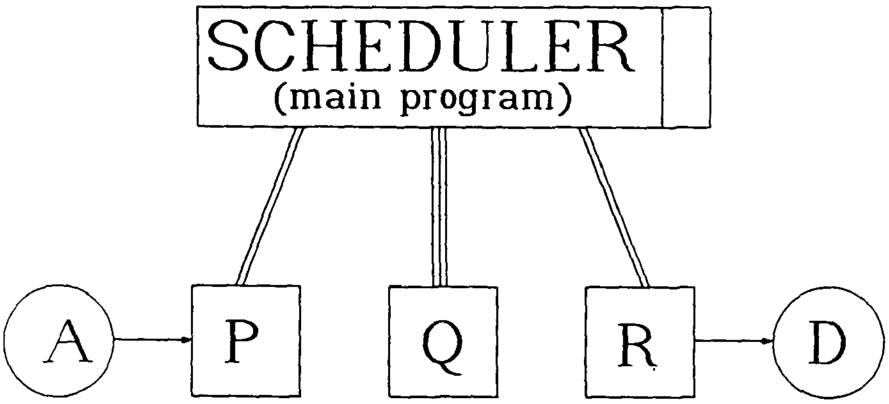


A.2 EXAMPLES OF PROGRAM INVERSION AND THE USE OF STATE VECTORS

To explain program inversion, refer to the specification flow diagram above. Program P reads file A and writes to file B. B is then read by program Q, which finally writes file C. If we invert P with respect to its input data stream, we could make P a subroutine of Q, so that whenever Q requires a record from file B it calls P. Conversely, Q could be made a subroutine of P, so that when P wants to write a record to file B, it calls Q. The diagram below shows an example of the inversion of a linear process sequence. Part (a) is a system specification diagram. Part (b) shows the inverted programs, and is termed a *system implementation* diagram.



a)



b)

In (b) above, data stream B is replaced by the connecting channel between program P and the Scheduler; data streams B and C are replaced by the channel between program Q and the Scheduler; finally, data stream C is replaced by the channel between program R and the Scheduler.

These transformations eliminate the need for actual intermediate files, and establish an automatic read/write sequence that is useful in system design. However, the program structure of the process continues to reflect the files' existence. The intermediate files, although nonexistent, contain an implicit structure reminiscent of the main files' structure.

Sections A.2.1 and A.2.2 show two examples of program inversion. The text pointer TXTPTR allows the program to always track exactly where it is within itself, and to return to that point when needed. Note that when inverted, the program becomes a *variable-state subroutine*.

A.2.1 Program Inversion with Respect to Its Input Data Stream

Before Inversion:

```
      PROGRAM WRFILE
           .
           .
           .
      declarations
           .
           .
           .
      OPEN (OUTPUT_FILE)
      OPEN (INTERMEDIATE_FILE)
101   CONTINUE
      READ (INTERMEDIATE_FILE) INVAR
      IF (INVAR .EQ. EOF) GO TO 201
           .
           .
           .
      process data
           .
           .
           .
      WRITE (OUTPUT_FILE) OUTVAR
      GO TO 101
201   CONTINUE
      STOP
      END
```

After Inversion

```
      SUBROUTINE WRFILE (INVAR)
           .
           .
           .
      declarations
           .
           .
           .
      DATA TXTPTR /1/
      GO TO (10001, 10002) TXTPTR
10001 CONTINUE
      OPEN (OUTPUT_FILE)
101   CONTINUE
      IF (INVAR .EQ. EOF) GO TO 201
           .
           .
           .
      process data
           .
           .
           .
      WRITE (OUTPUT_FILE) OUTVAR
      TXTPTR = 2
      RETURN
10002 CONTINUE
      GO TO 101
201   CONTINUE
      RETURN
      END
```

A.2.2 Program Inversion with Respect to Its Output Data Stream

Before Inversion:

```
      PROGRAM RDFILE
          .
          .
          .
      declarations
          .
          .
          .
      OPEN (INPUT_FILE)
      OPEN (INTERMEDIATE_FILE)
101   CONTINUE
      READ (INPUT_FILE) INVAR
      IF (INVAR .EQ. EOF) GO TO 201
          .
          .
          .
      process data
          .
          .
          .
      WRITE (INTERMEDIATE_FILE) OUTVAR
      GO TO 101
201   CONTINUE
      STOP
      END
```

After Inversion

```
      SUBROUTINE RDFILE (INVAR)
          .
          .
          .
      declarations
          .
          .
          .
      DATA TXTPTR /1/
      GO TO (10001, 10002) TXTPTR
10001 CONTINUE
      OPEN (INPUT_FILE)
101   CONTINUE
      READ (INPUT_FILE) INVAR
      IF (INVAR .EQ. EOF) GO TO 201
          .
          .
          .
      process data
          .
          .
          .
      TXTPTR = 2
      RETURN
10002 CONTINUE
      GO TO 101
201   CONTINUE
      RETURN
      END
```

A.3 REFERENCE AND BIBLIOGRAPHY

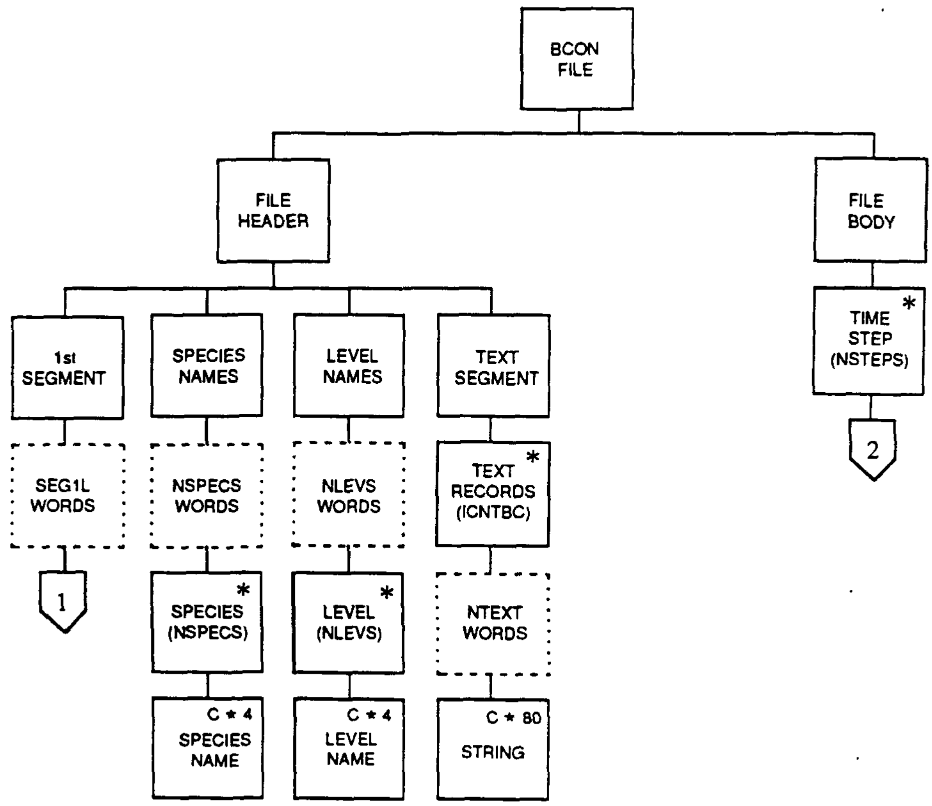
Jackson, M. A., 1975. *Principles of Program Design.* A.P.I.C. Studies in Data Processing 12. Academic Press, London, United Kingdom. 299 pp.

Jackson, M. A., 1983. *System Development.* Prentice-Hall International Series in Computer Science. Prentice/Hall International, Englewood Cliffs, New Jersey. 418 pp.

APPENDIX B

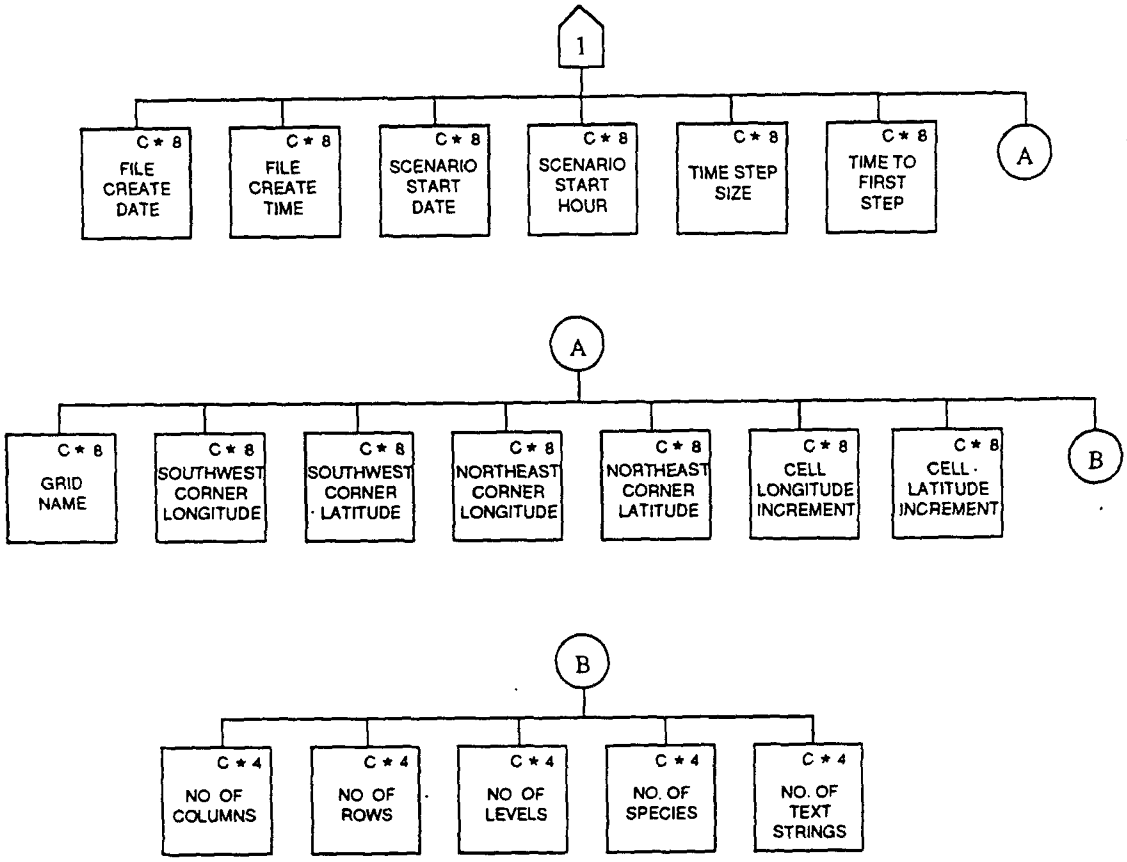
DESIGN AND STRUCTURE DIAGRAMS FOR THE ROM2.1 DATA FILES IN THE ROMNET REGION

Refer to Appendix A for the explanation of symbols used.
Also note that boxes drawn with broken lines denote logical records.

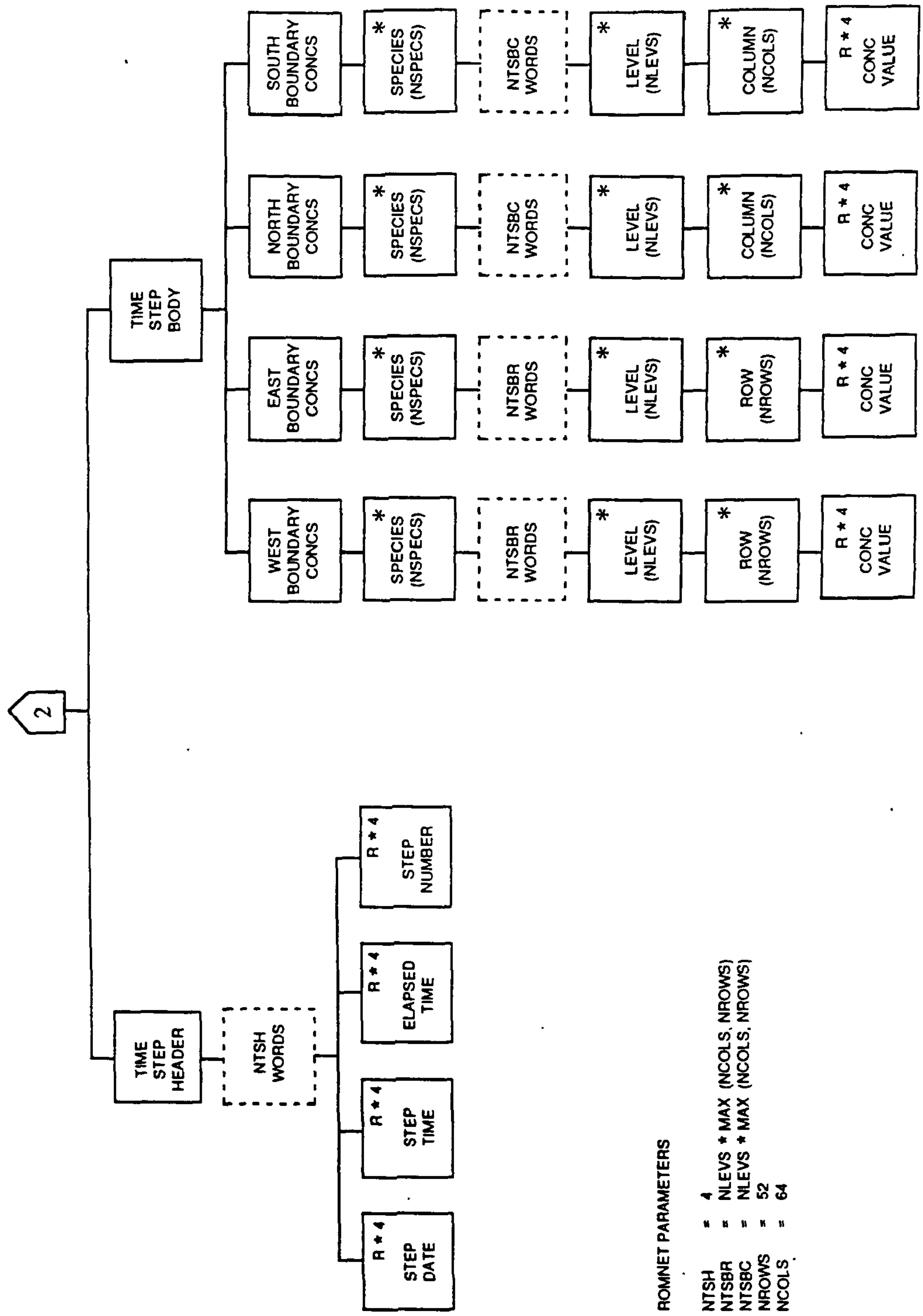


SEG1L = (19 *2) + (5* 1)
NSPECS = 35
NLEVS = 3
ICNTBC ≤ 20
NTEXT = 20
NSTEPS < ∞

Figure B-1. The BCON file (page 1 of 3)



The BCON file (page 2 of 3)



The BCON file (page 3 of 3)

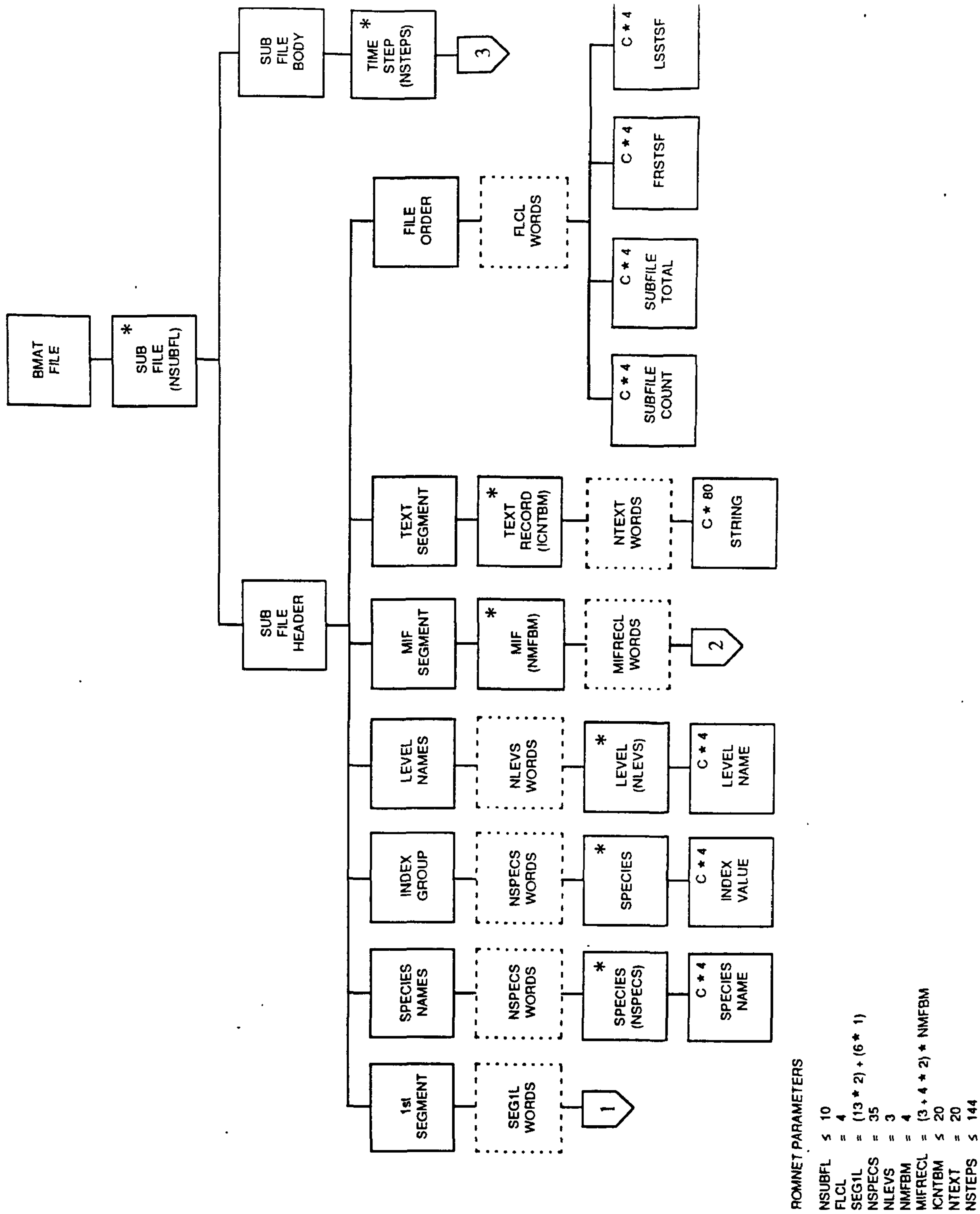
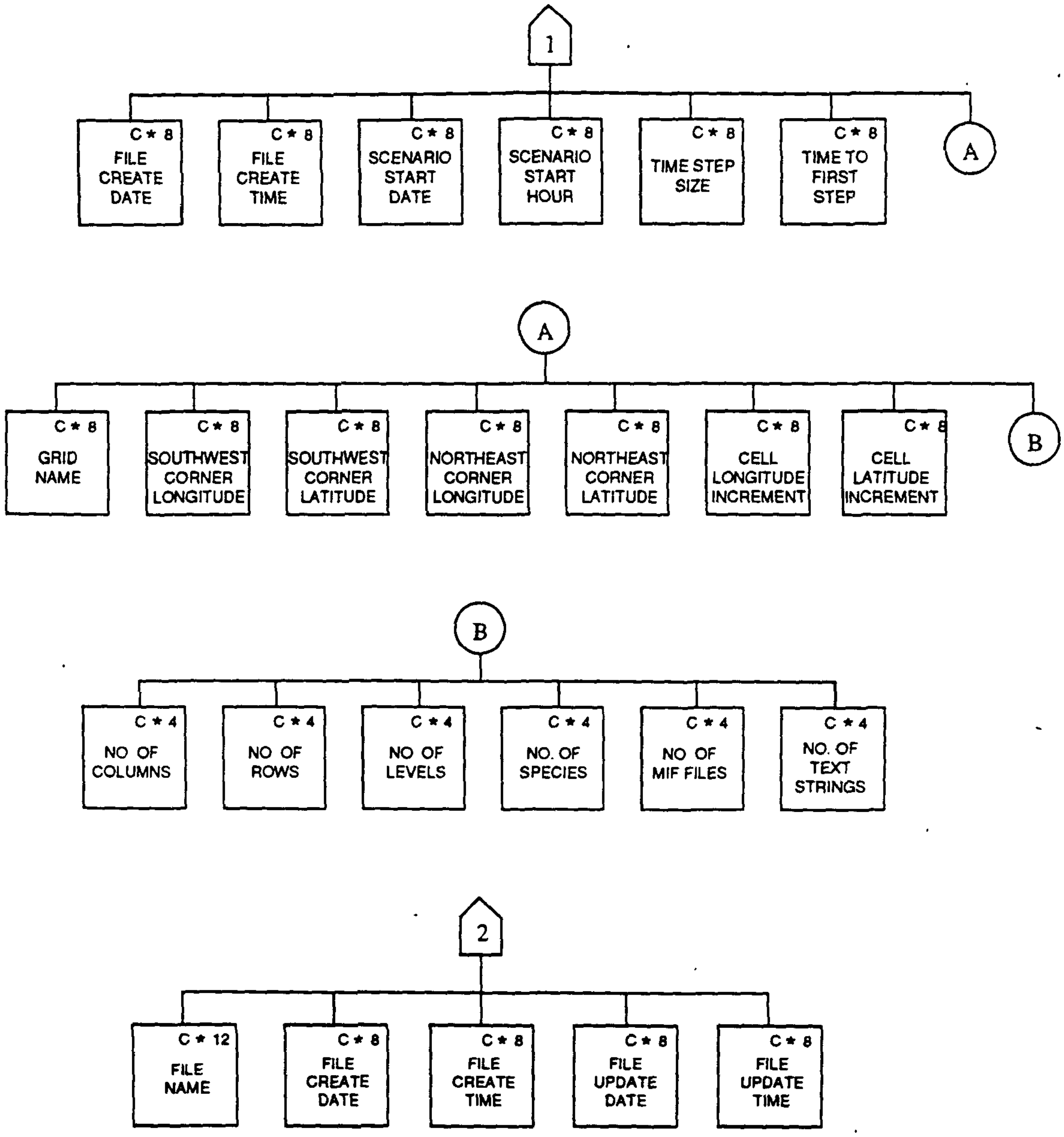
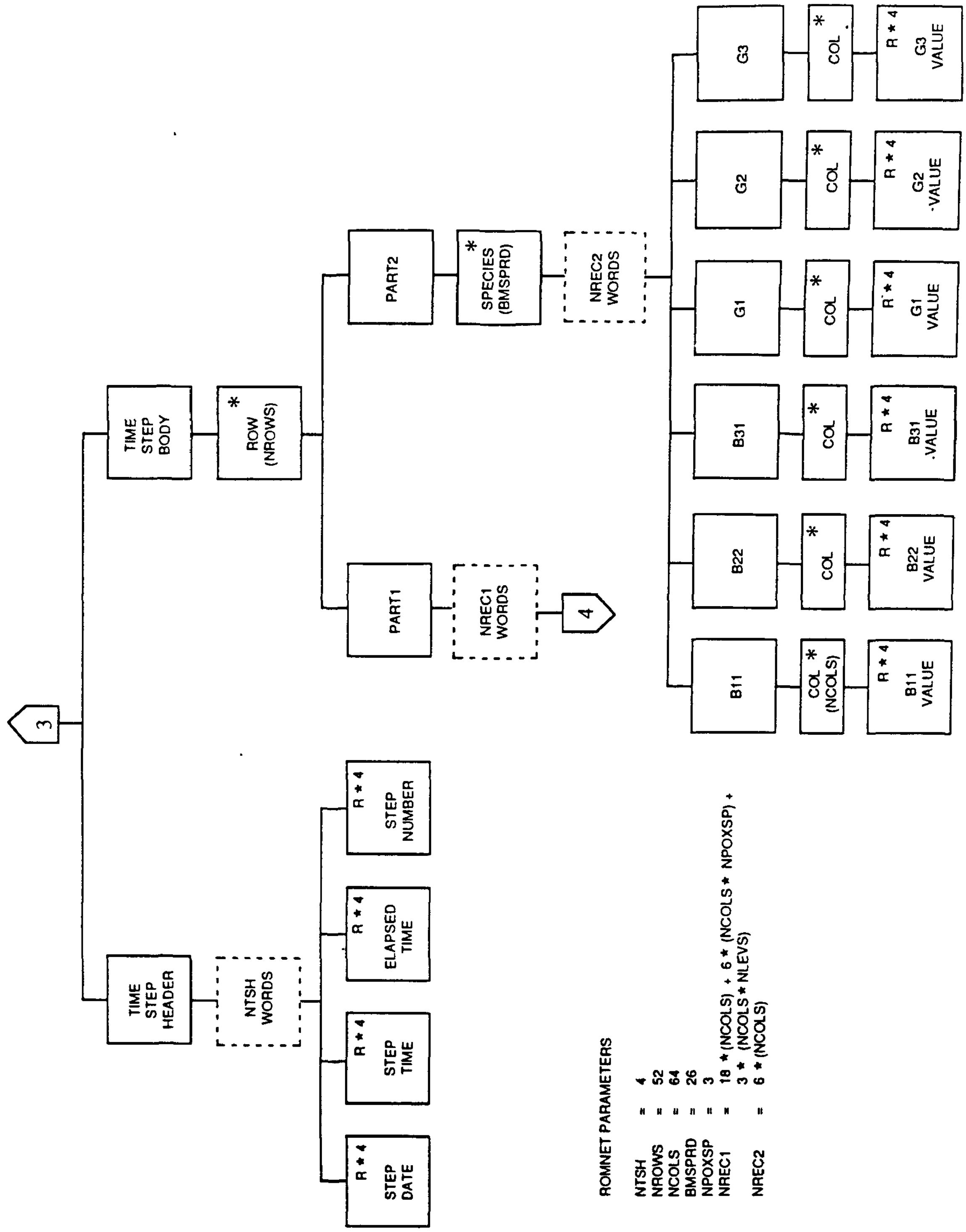


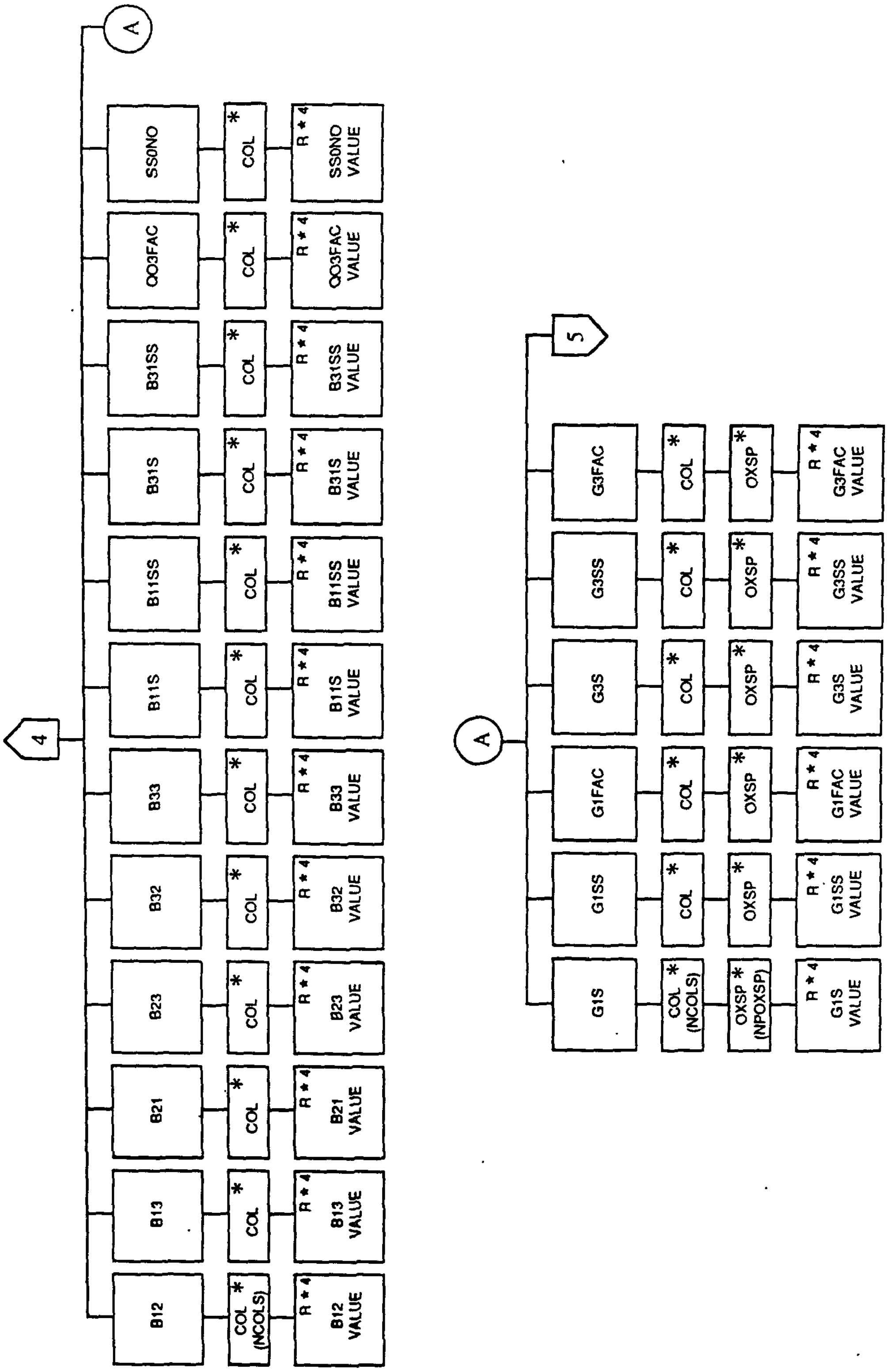
Figure B-2. The BMAT file (page 1 of 5)



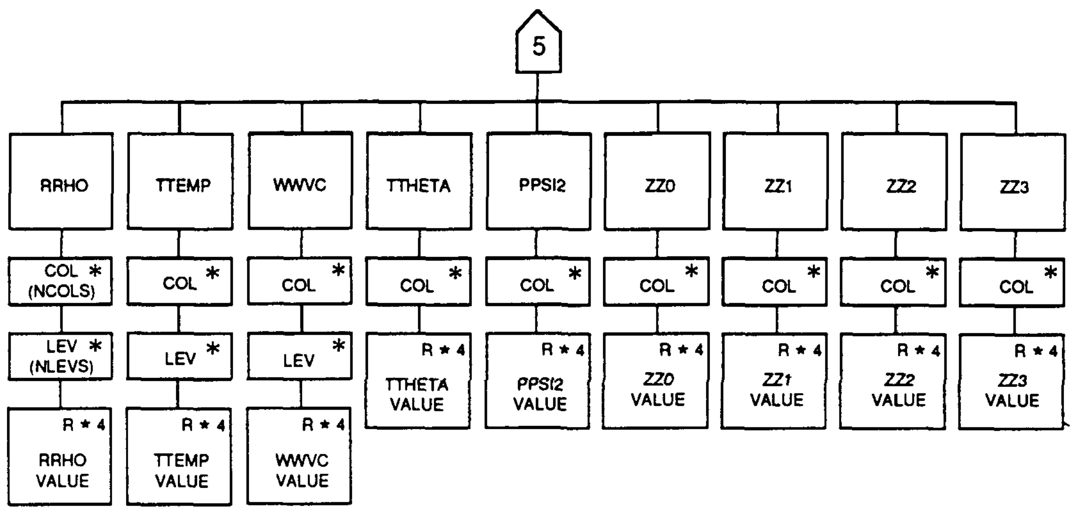
The BMAT file (page 2 of 5)



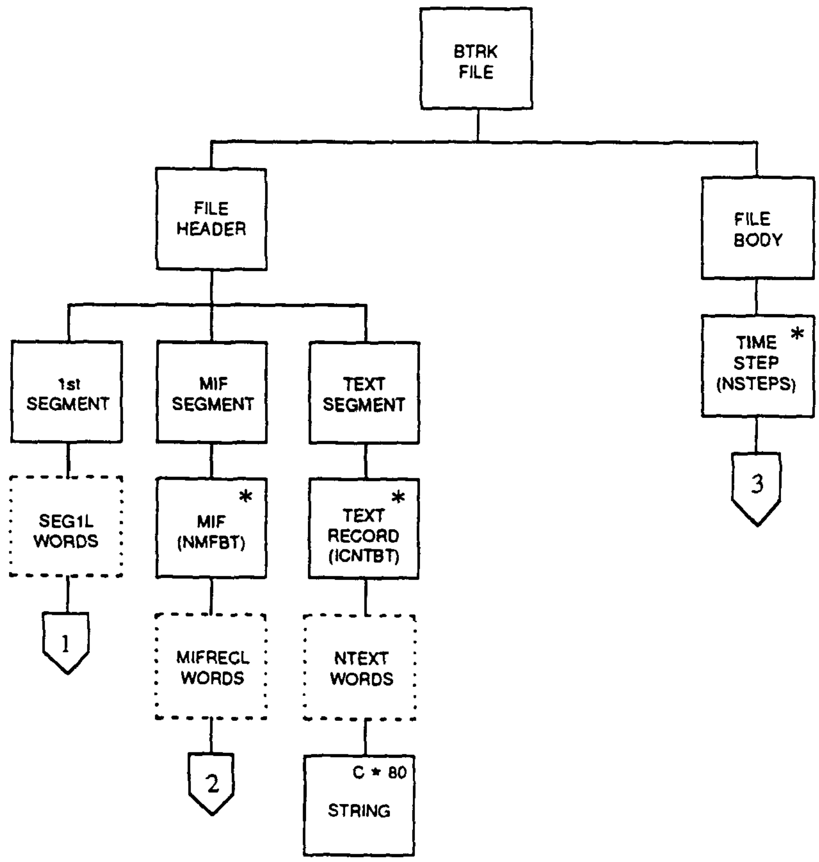
The BMAT file (page 3 of 5)



The BMAT file (page 4 of 5)

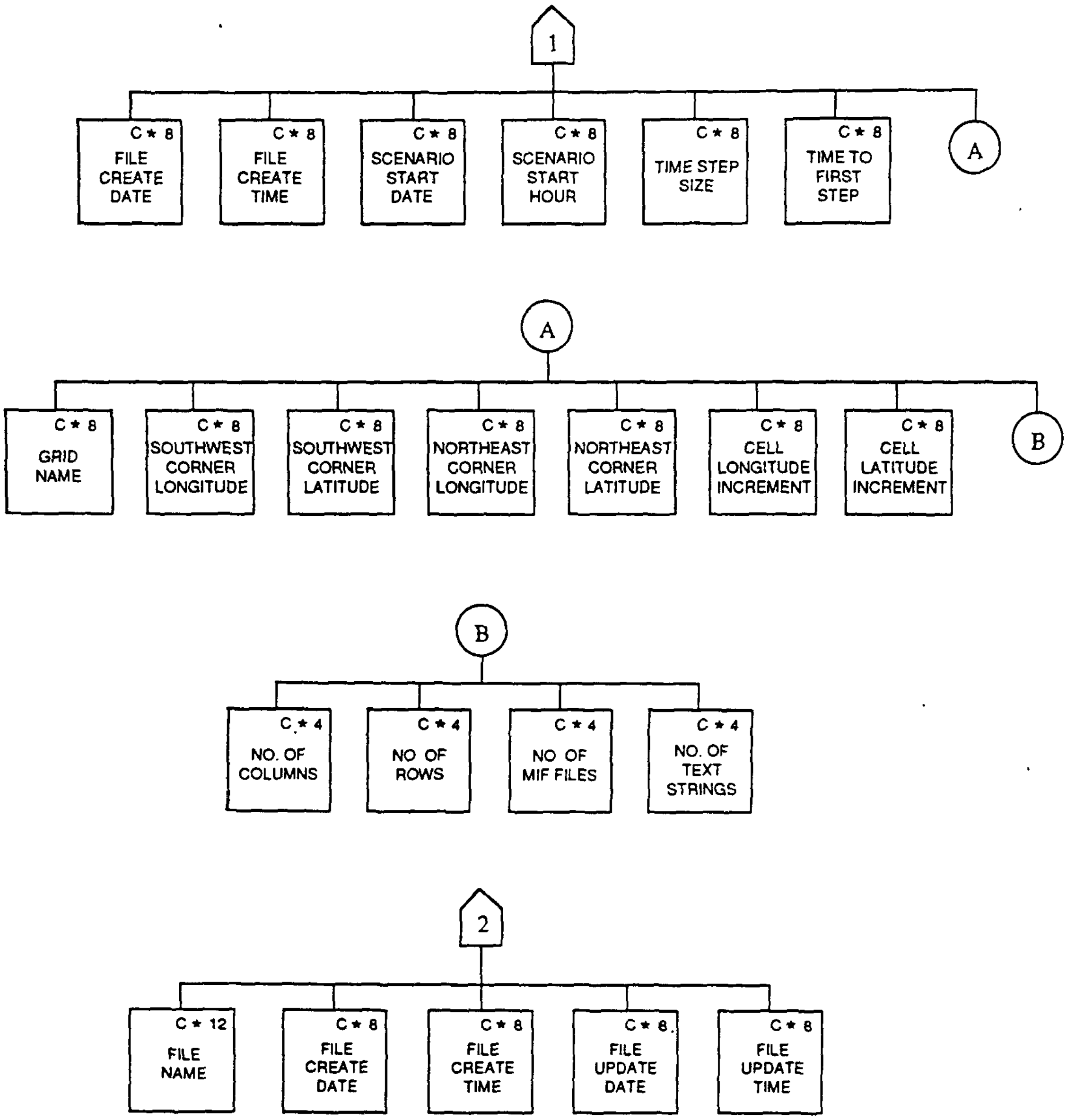


The BMAT file (page 5 of 5)

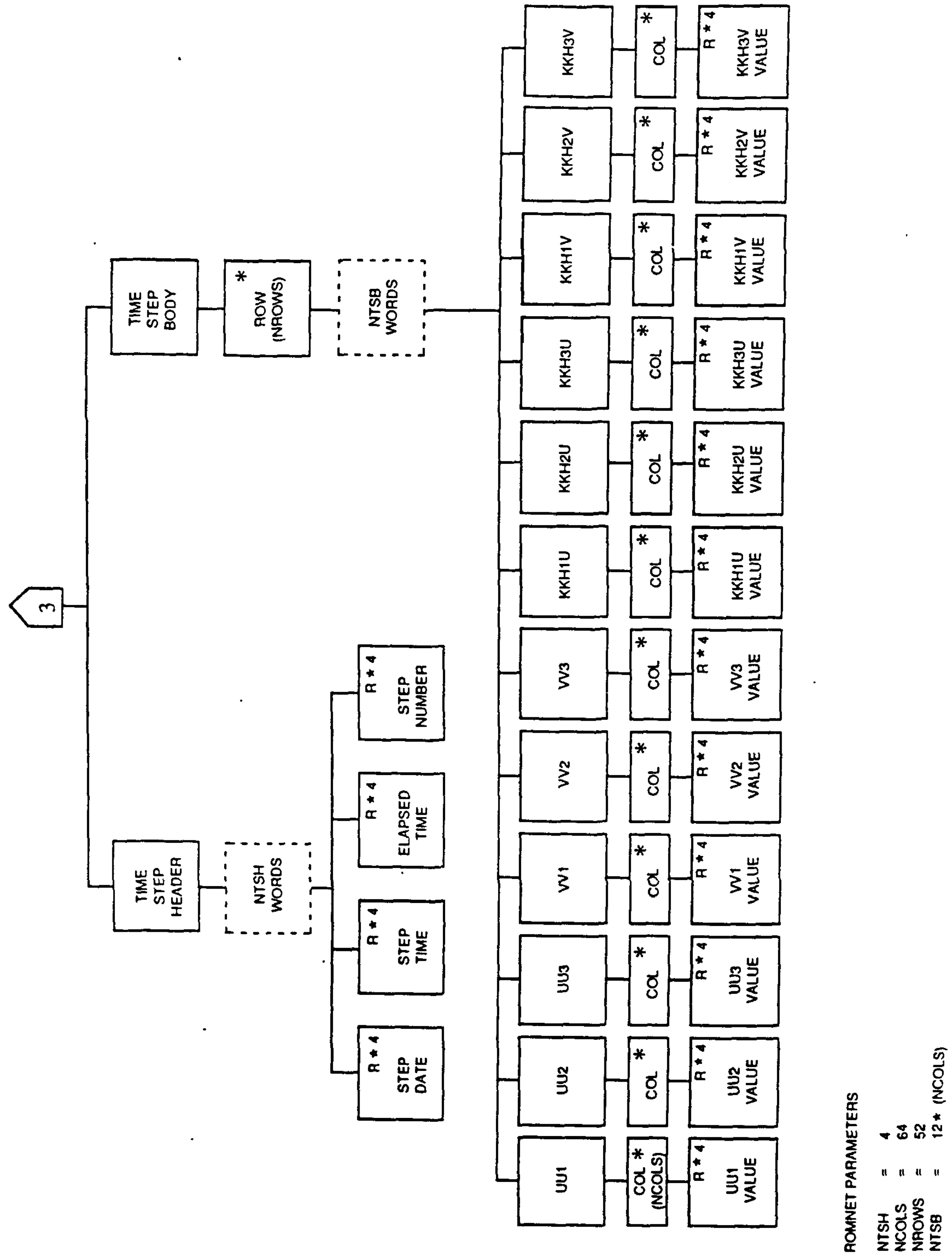


SEG1L = (13 * 2) + (4 * 1)
NMFBT = 6
MIFRECL = (3 + 4 * 2) * NMFBT
ICNTBT ≤ 20
NTEXT = 20
NSTEPS ≤ 144

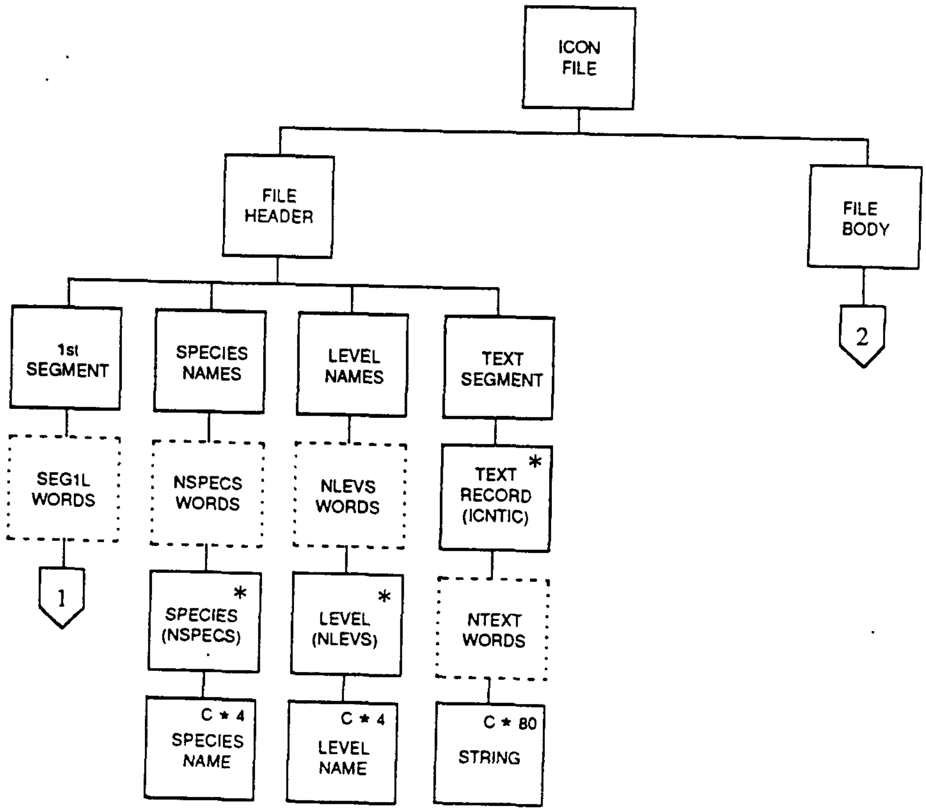
Figure B-3. The BTRK file (page 1 of 3)



The BTRK file (page 2 of 3)

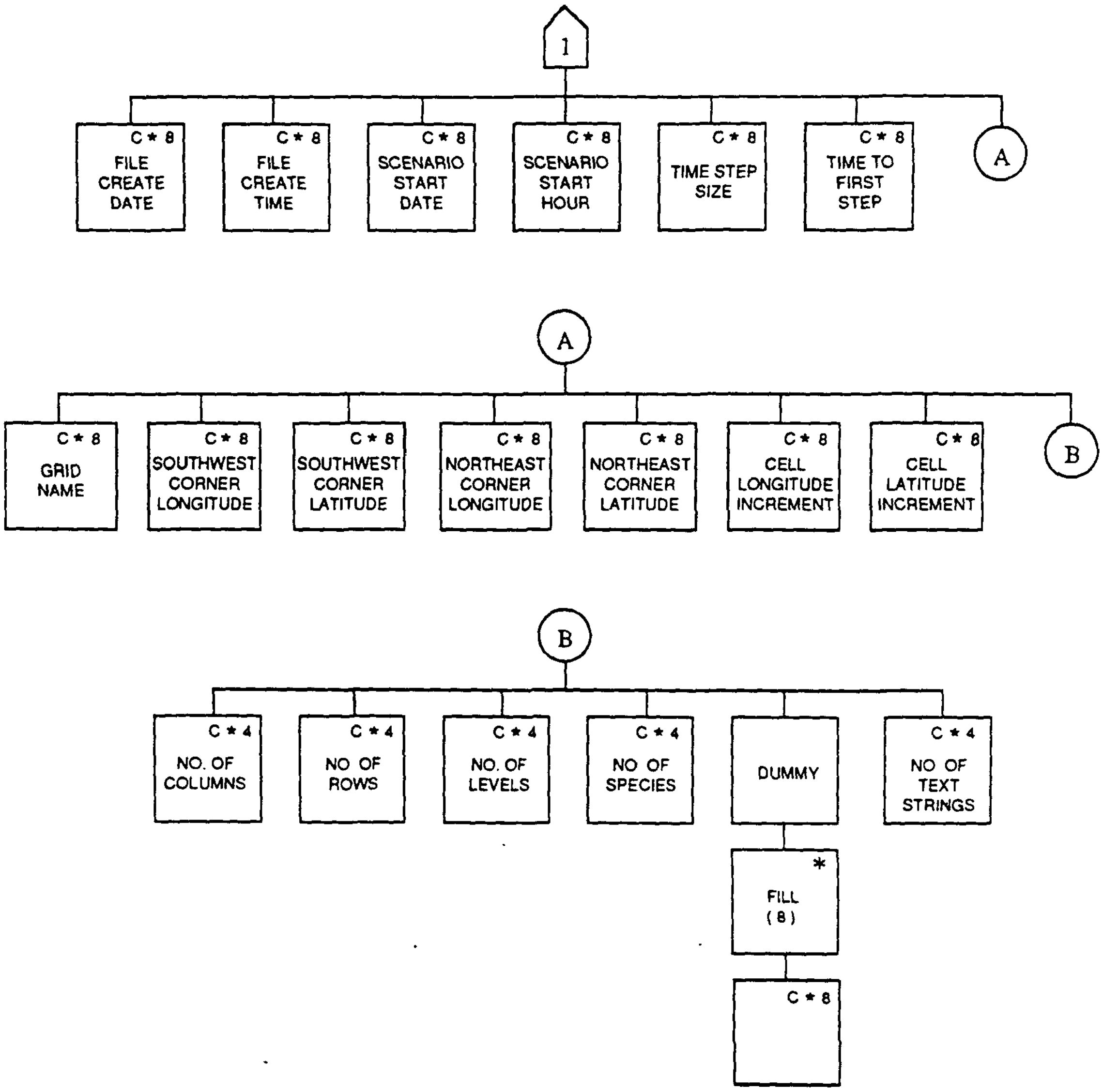


The BTRK file (page 3 of 3)

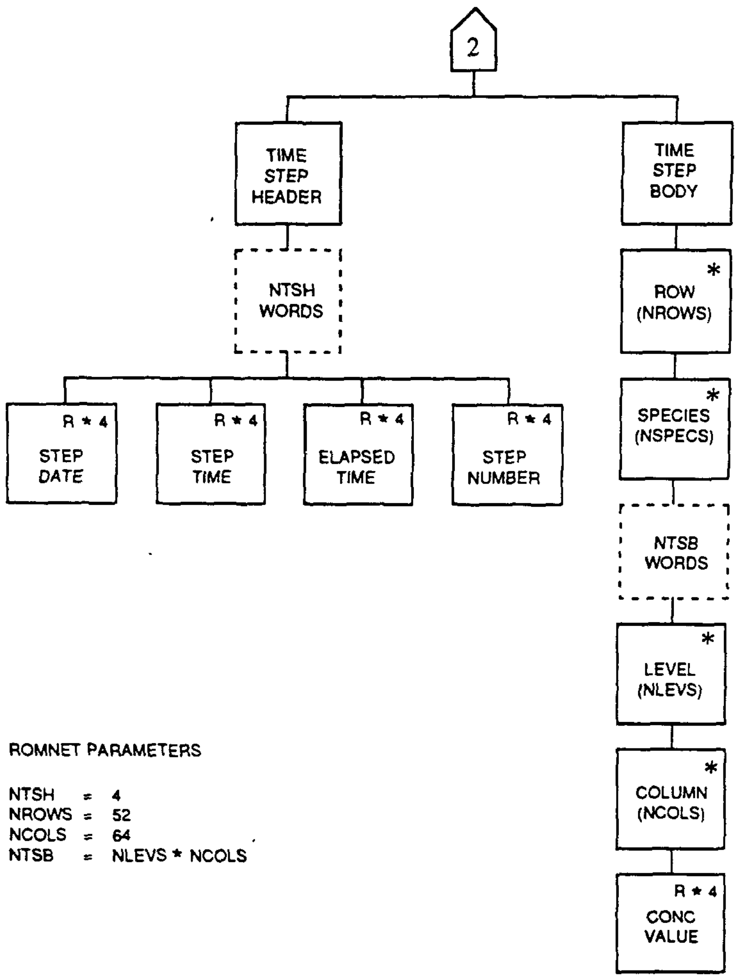


SEG1L	=	(21 * 2) + (5 * 1)
NSPECS	=	35
NLEVS	=	3
ICNTIC	≤	20
NTEXT	=	20

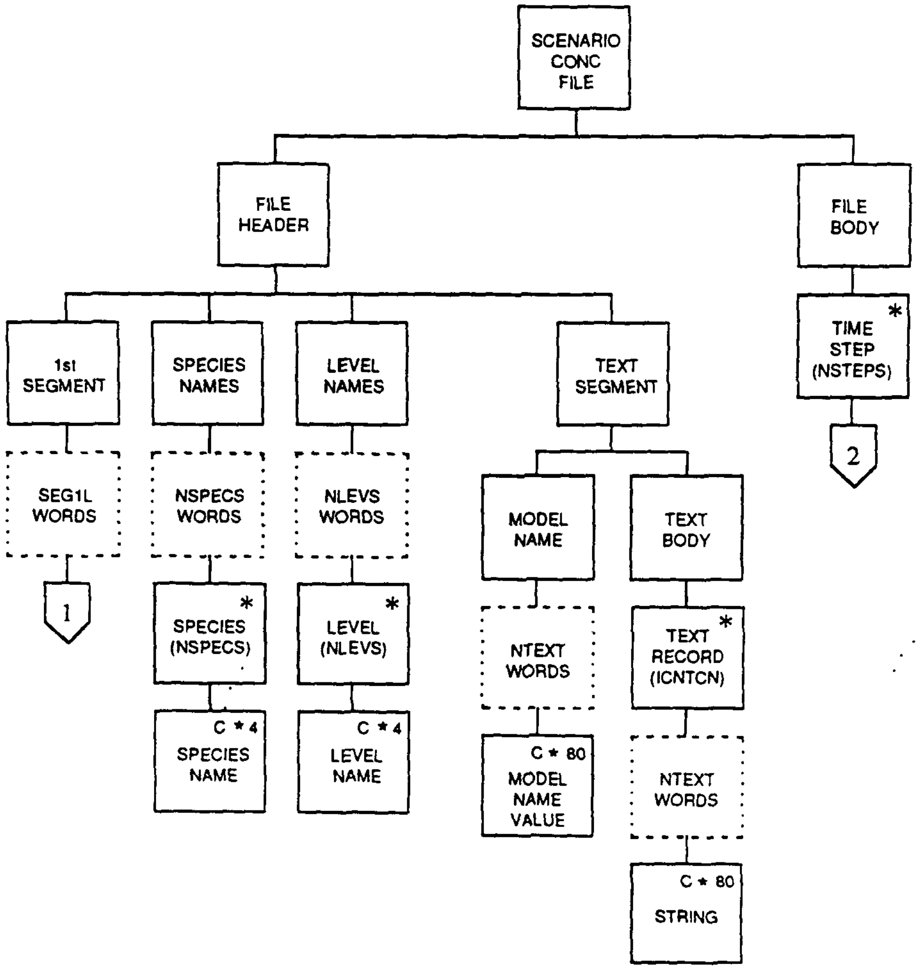
Figure B-4. The ICON file (page 1 of 3)



The ICON file (page 2 of 3)

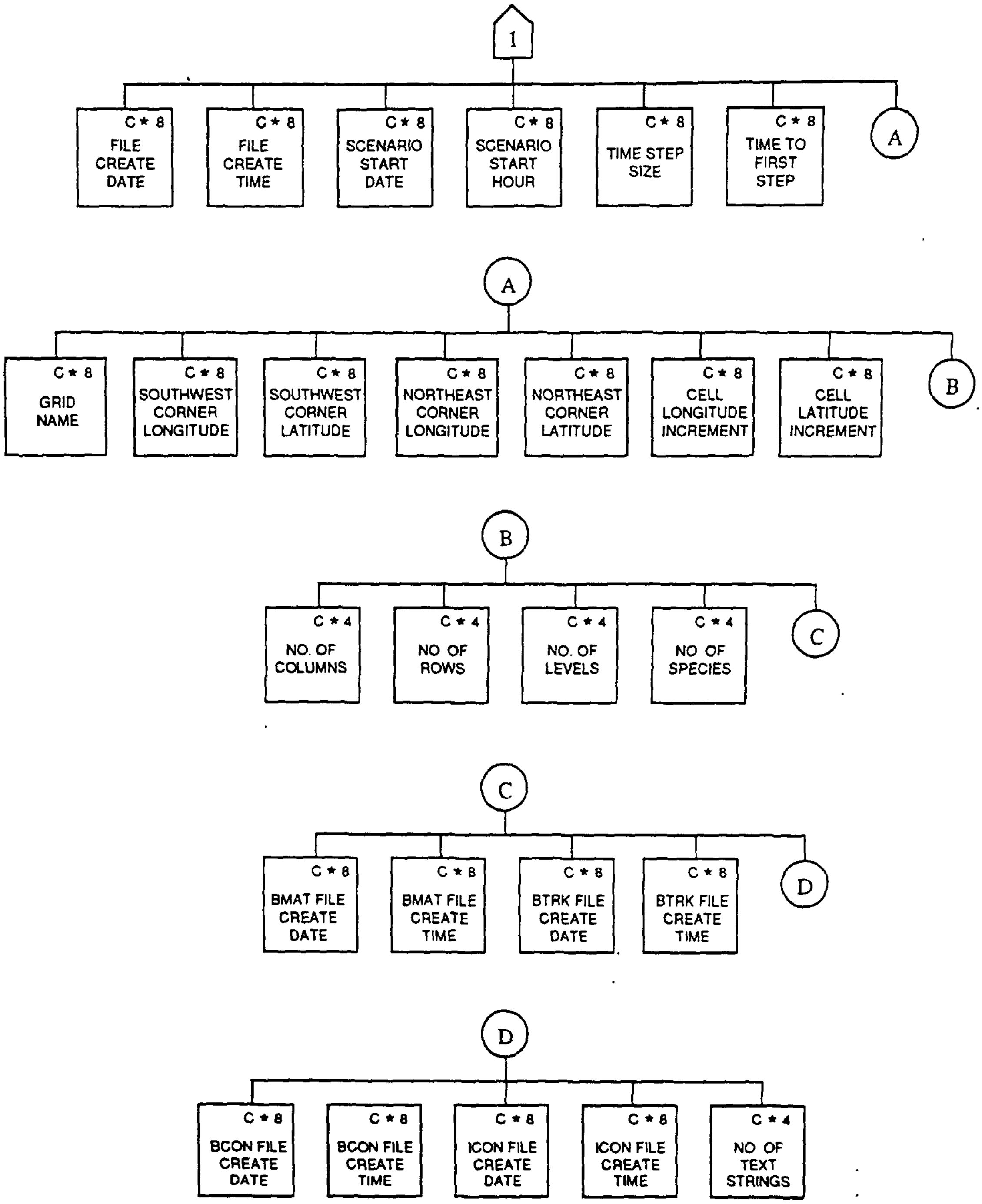


The ICON file (page 3 of 3)

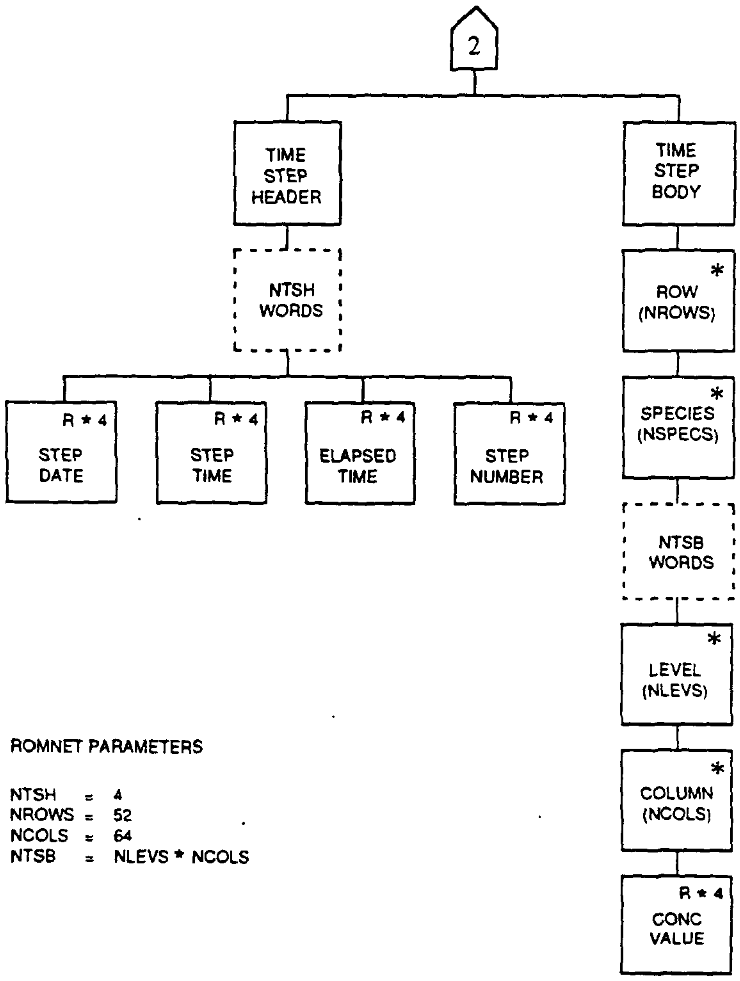


SEG1L = (21 * 2) + (5 * 1)
NSPECS = 35
NLEVS = 3
ICNTCN ≤ 20
NTEXT = 20
NSTEPS ≤ 145

Figure B-5. The CONC file (page 1 of 3)



The CONC file (page 2 of 3)



The CONC file (page 3 of 3)

APPENDIX C

DESIGN AND STRUCTURE DIAGRAMS FOR THE PRINCIPAL ROM2.1 SUBROUTINES

Refer to Appendix A for the explanation of symbols used.
Subprograms are indicated in italics.
T.S.H. = Time Step Header

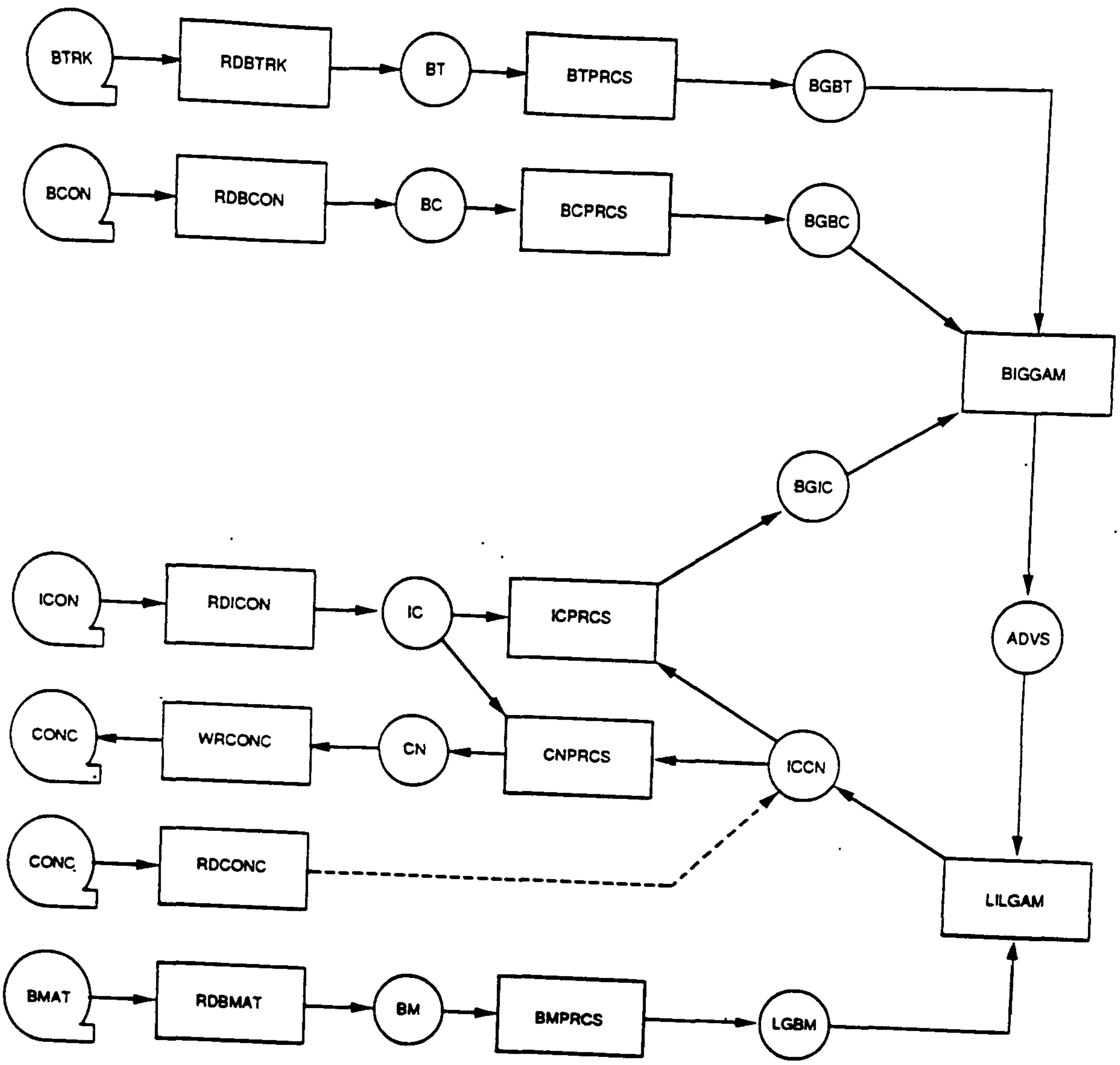


Figure C-1. Core Model system specification diagram.

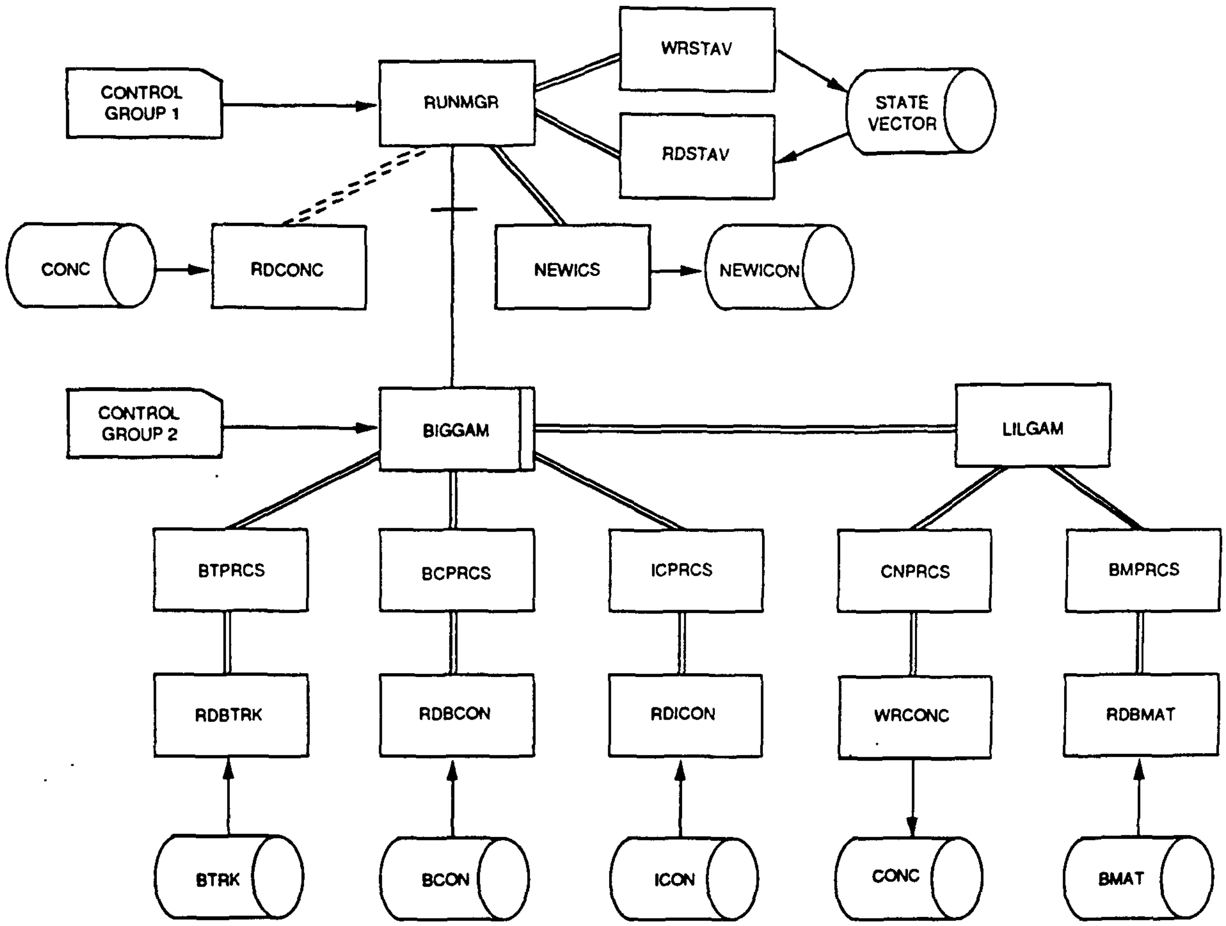


Figure C-2. Core Model Jackson Structured Design implementation diagram.

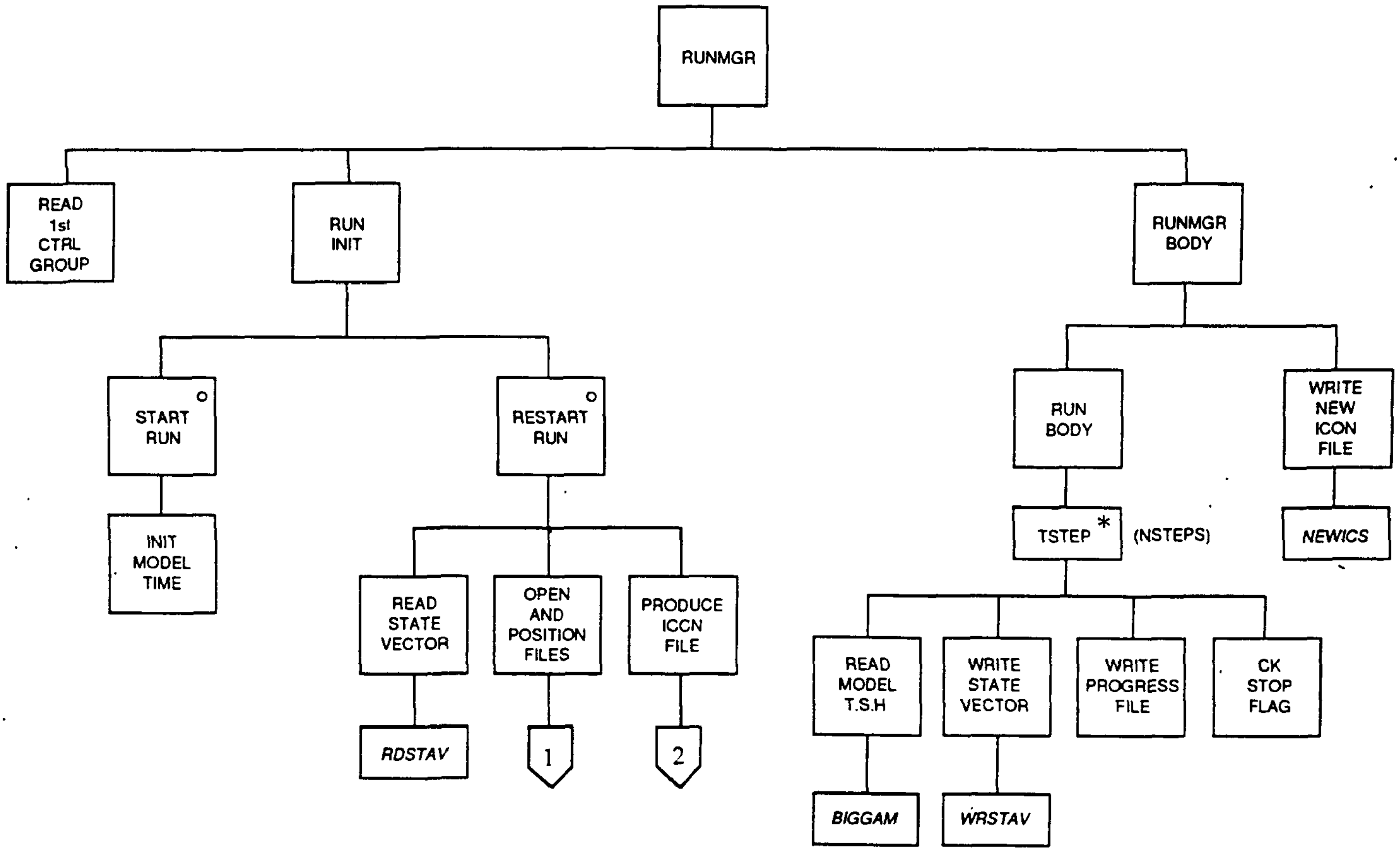
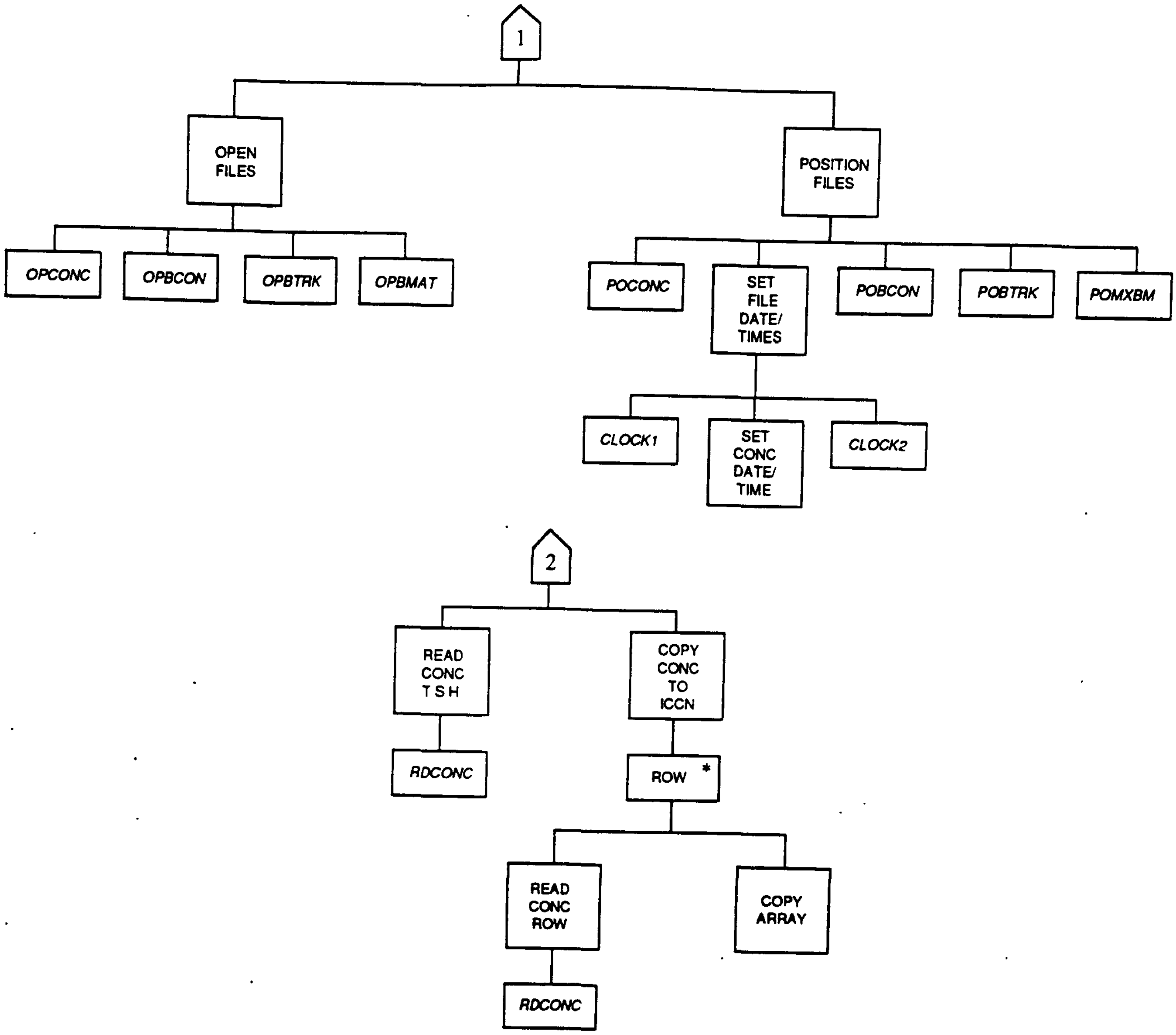


Figure C-3. RUNMGR (page 1 of 2).



RUNMGR (page 2 of 2).

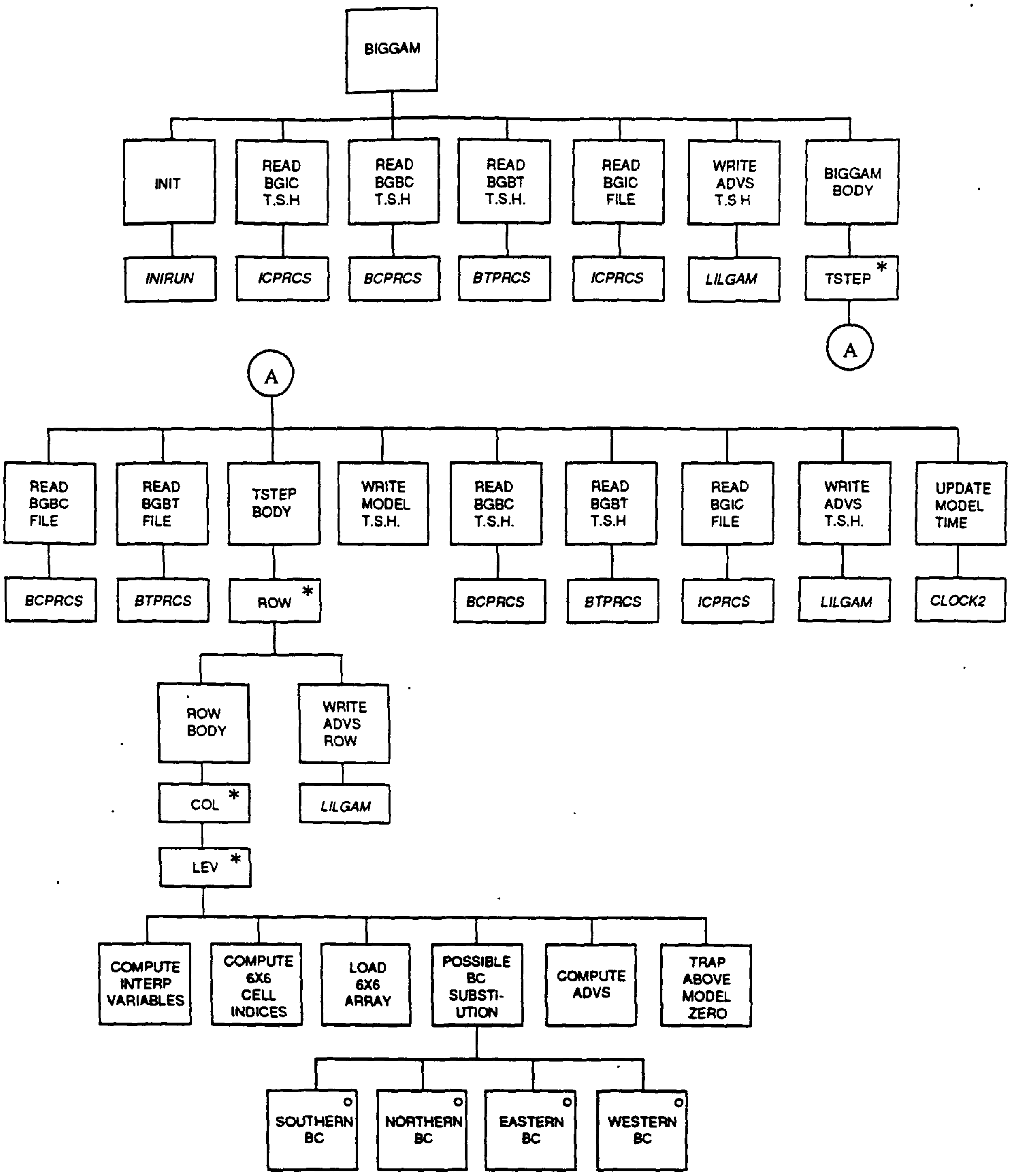


Figure C-4. BIGGAM (page 1 of 1).

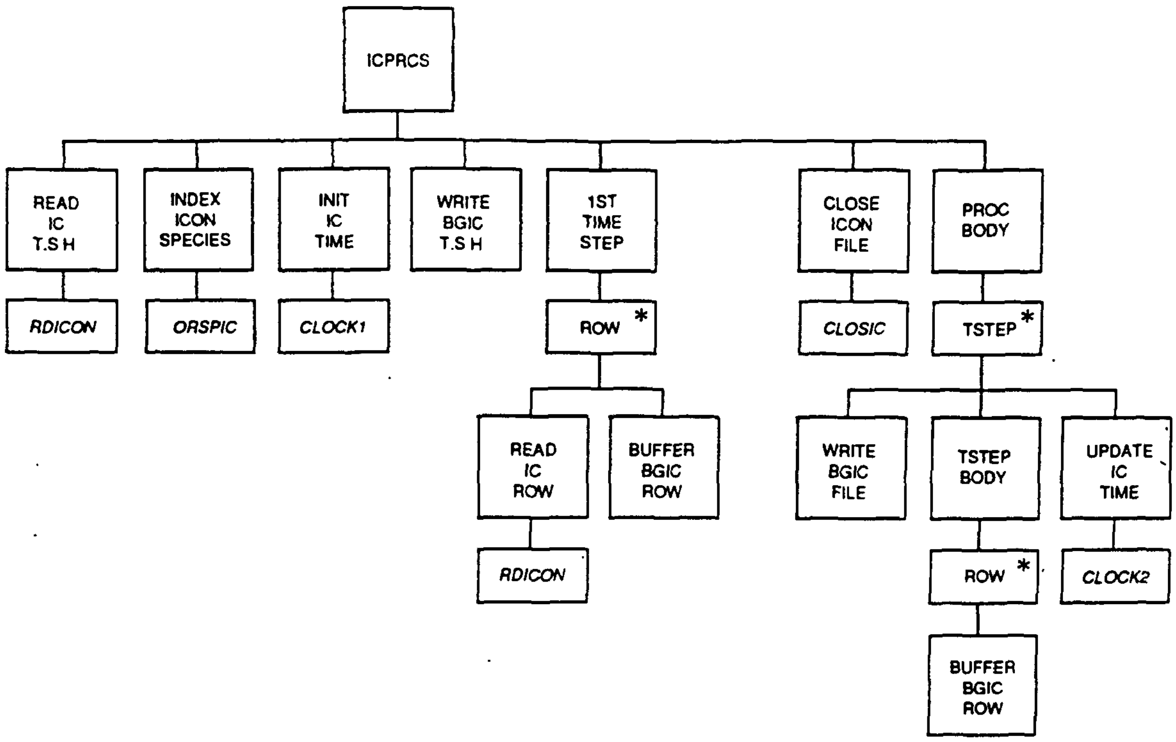


Figure C-5. ICPRCS (page 1 of 1).

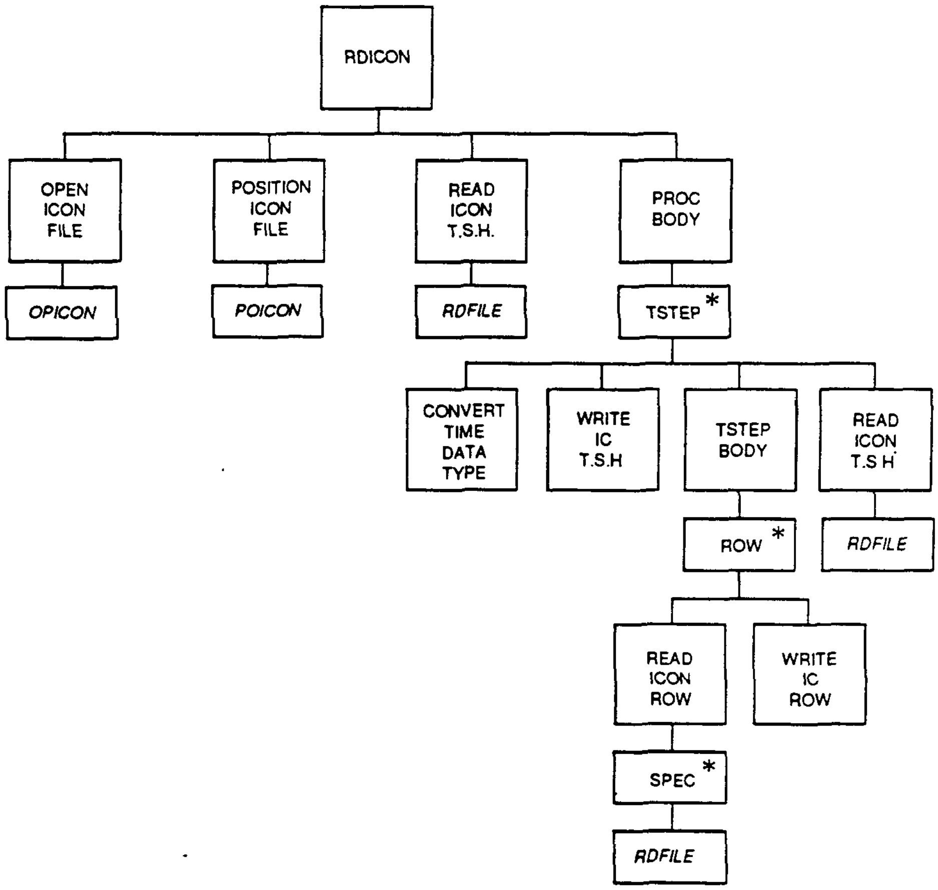


Figure C-6. RDICON (page 1 of 1).

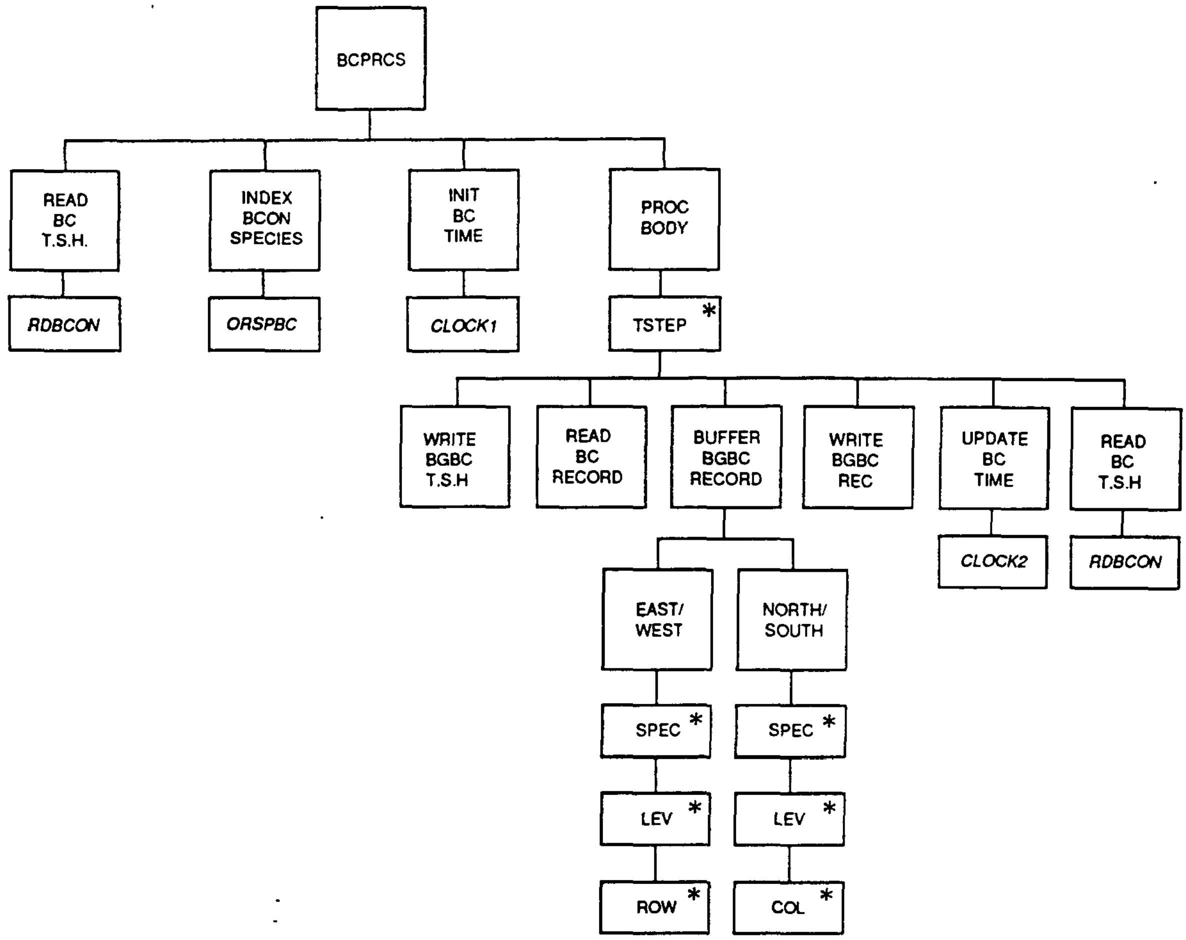


Figure C-7. BCPRCS (page 1 of 1).

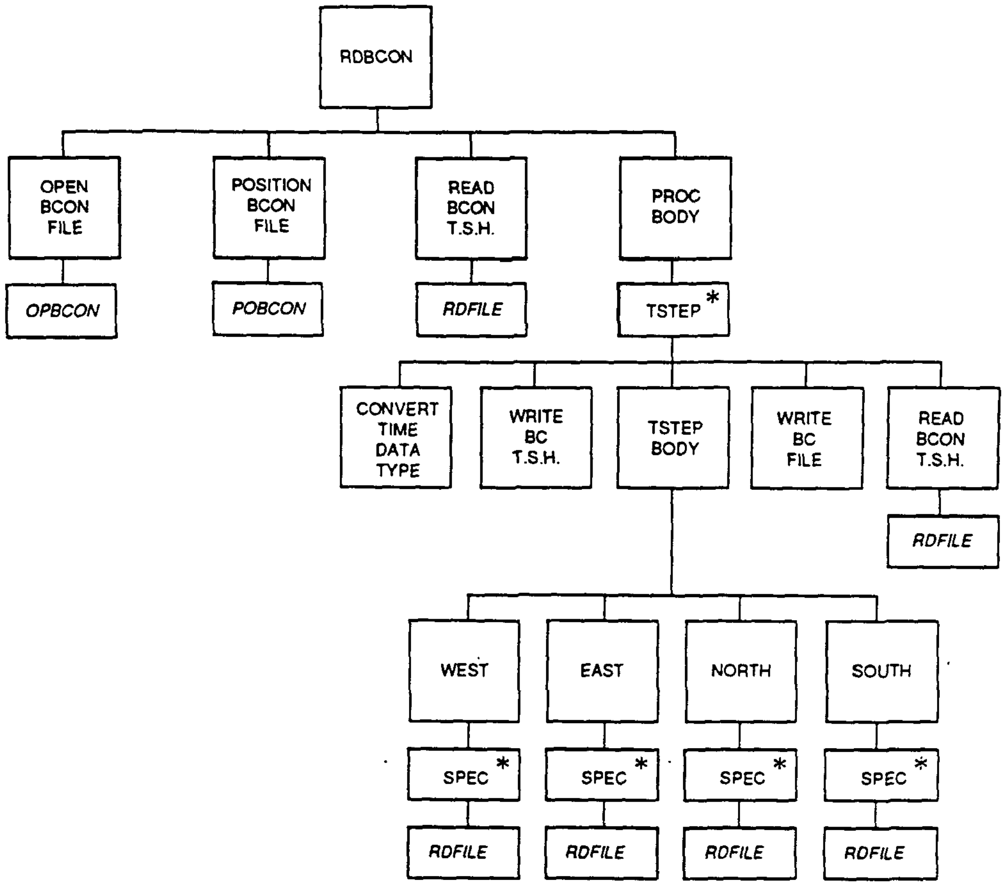


Figure C-8. RDBCON (page 1 of 1).

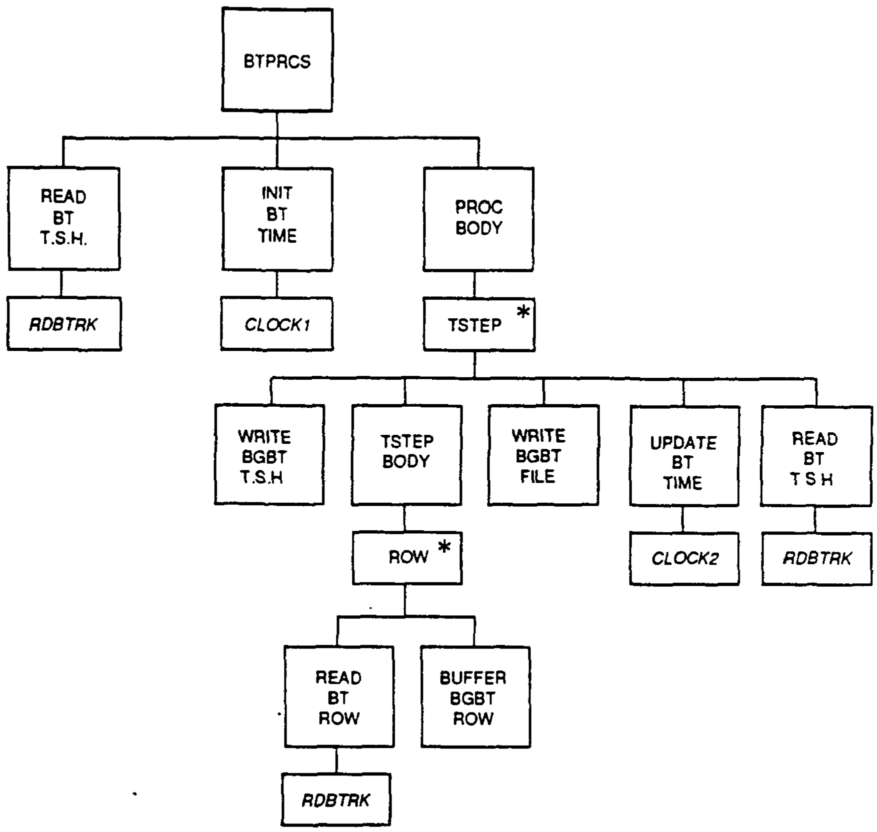


Figure C-9. BTPRCS (page 1 of 1).

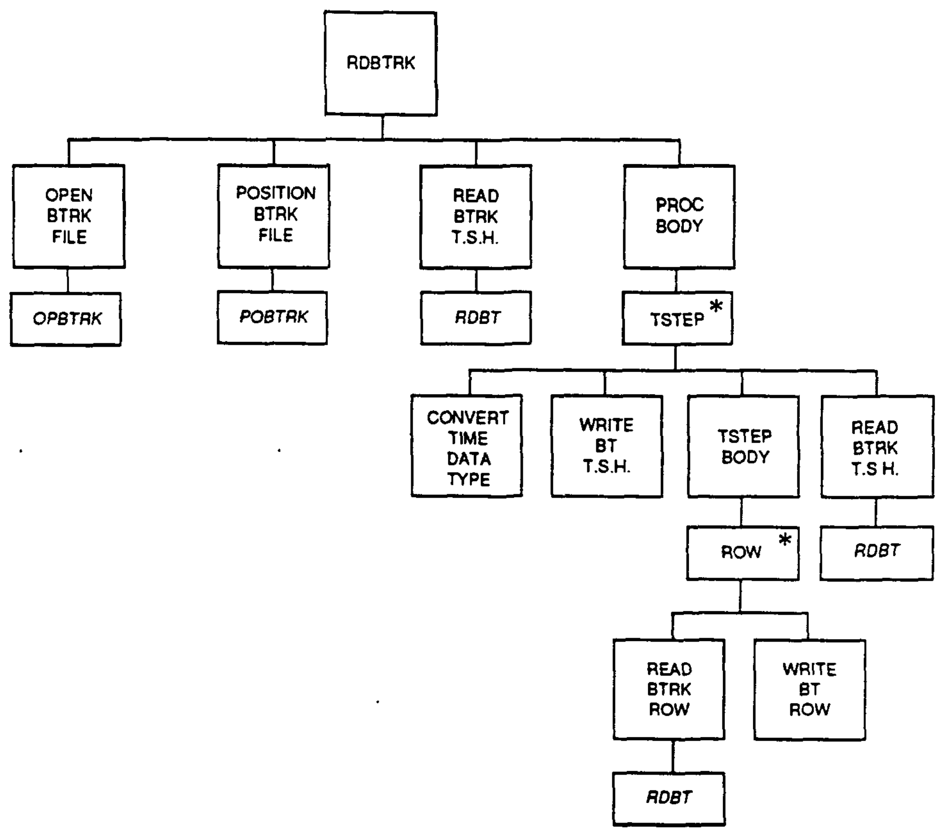


Figure C-10. RDBTRK (page 1 of 1).

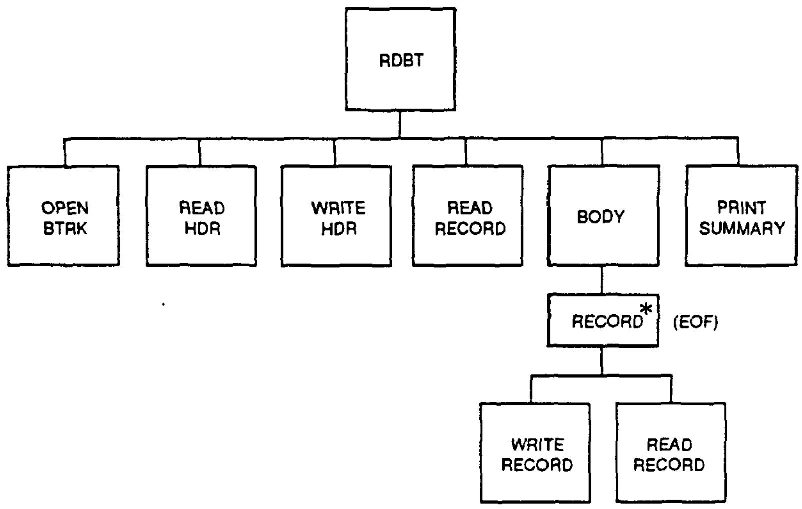


Figure C-11. RDBT (page 1 of 1).

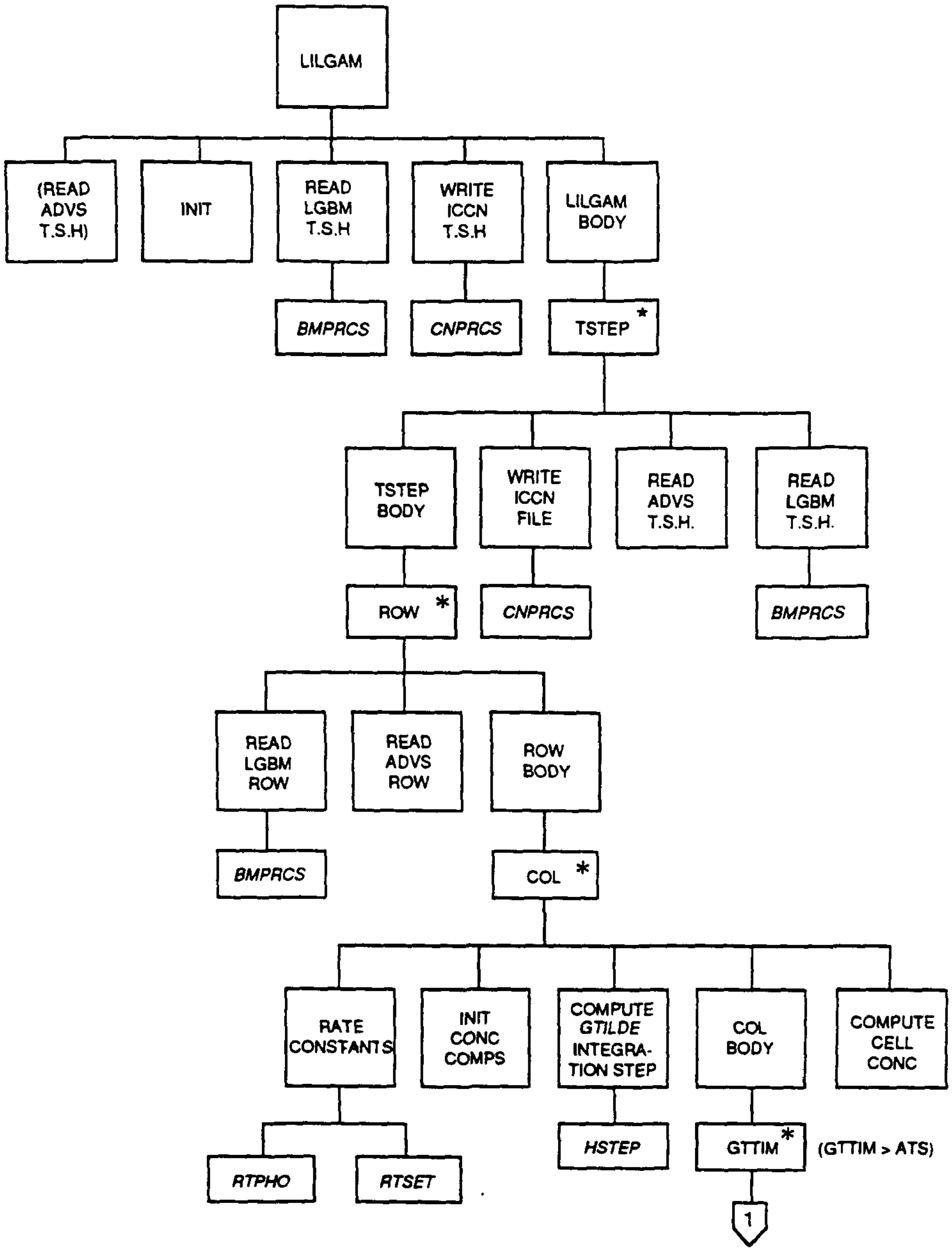
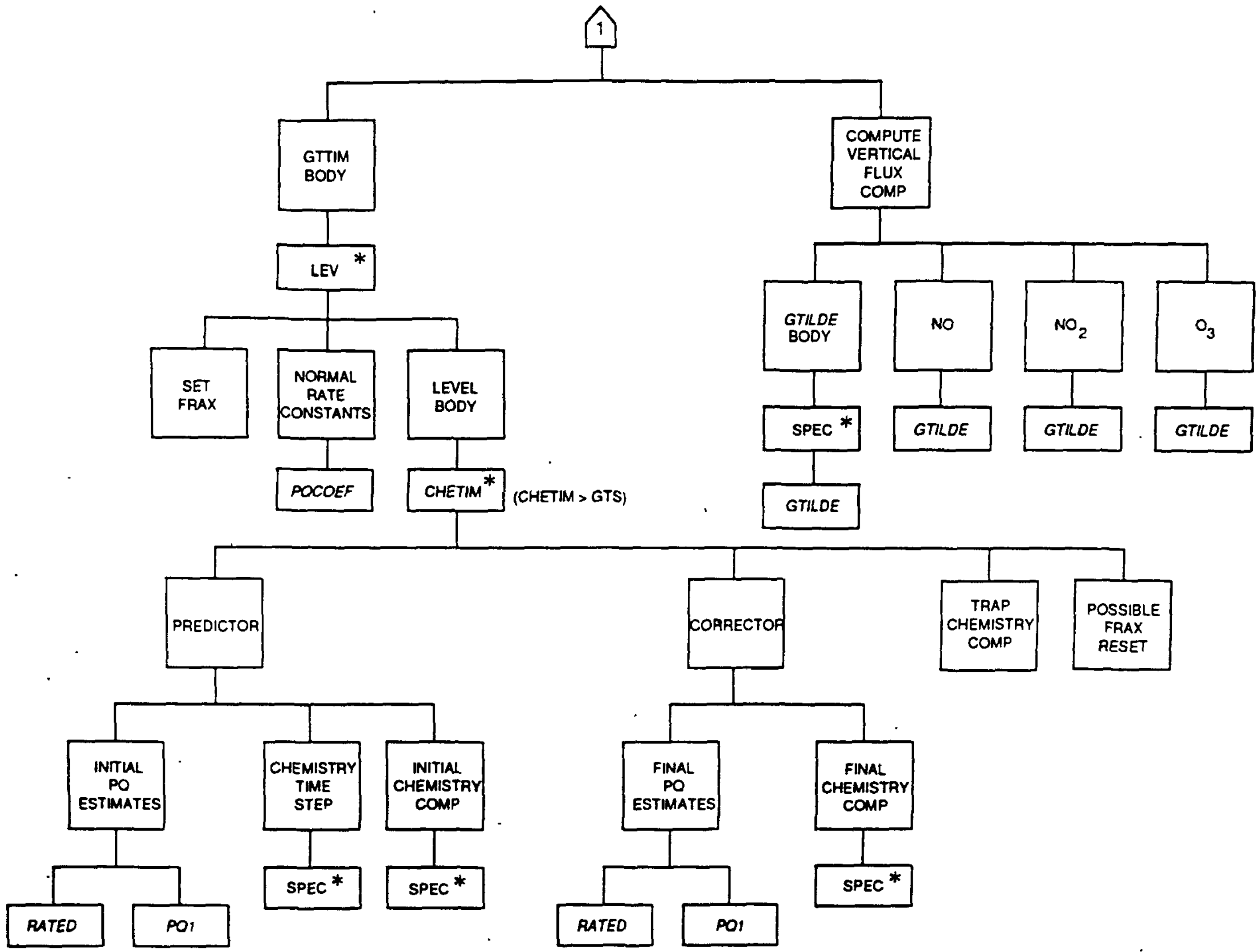


Figure C-12. LILGAM (page 1 of 2).



LILGAM (page 2 of 2).

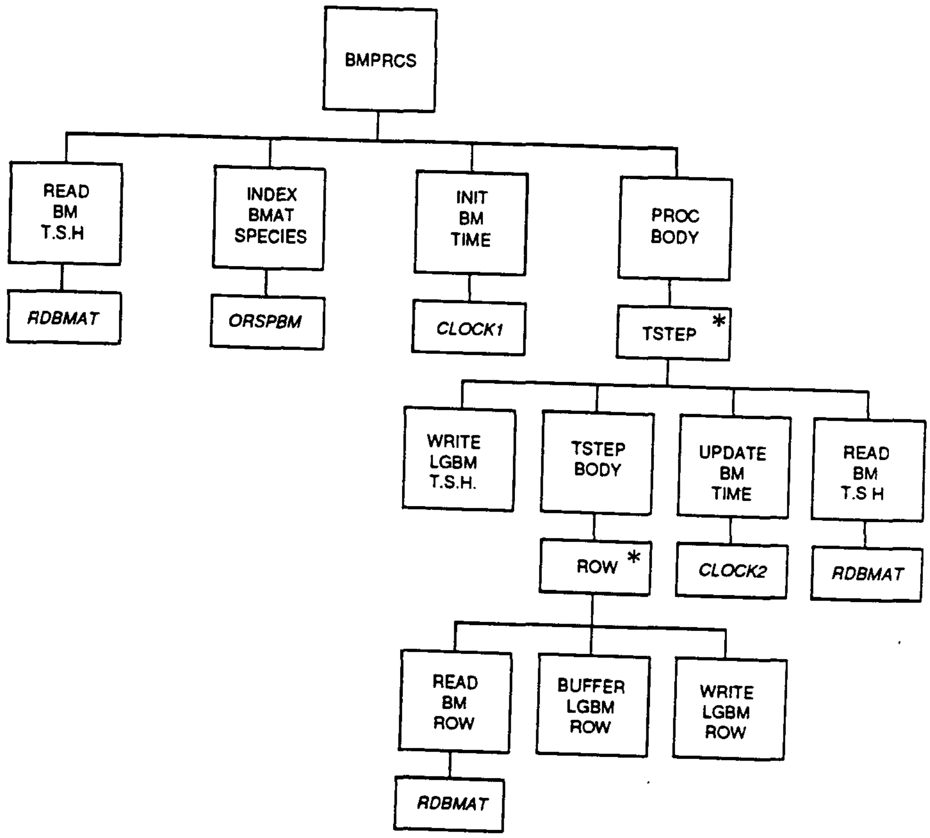


Figure C-13. BMPRCS (page 1 of 1).

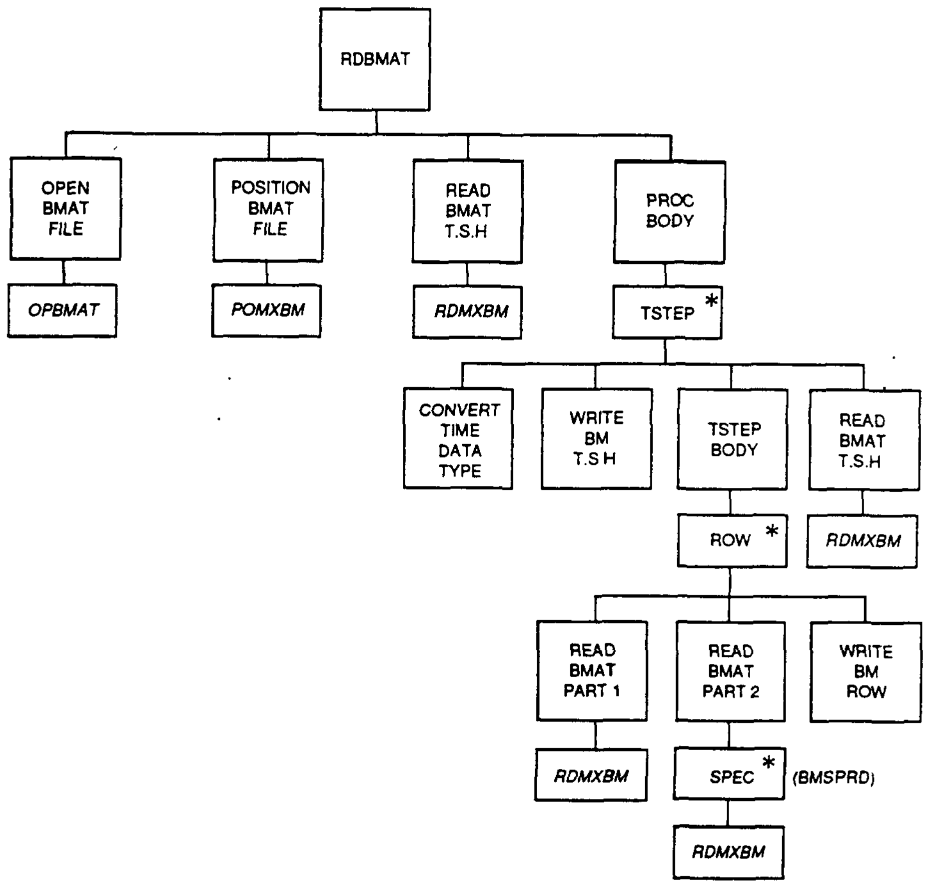


Figure C-14. RDBMAT (page 1 of 1).

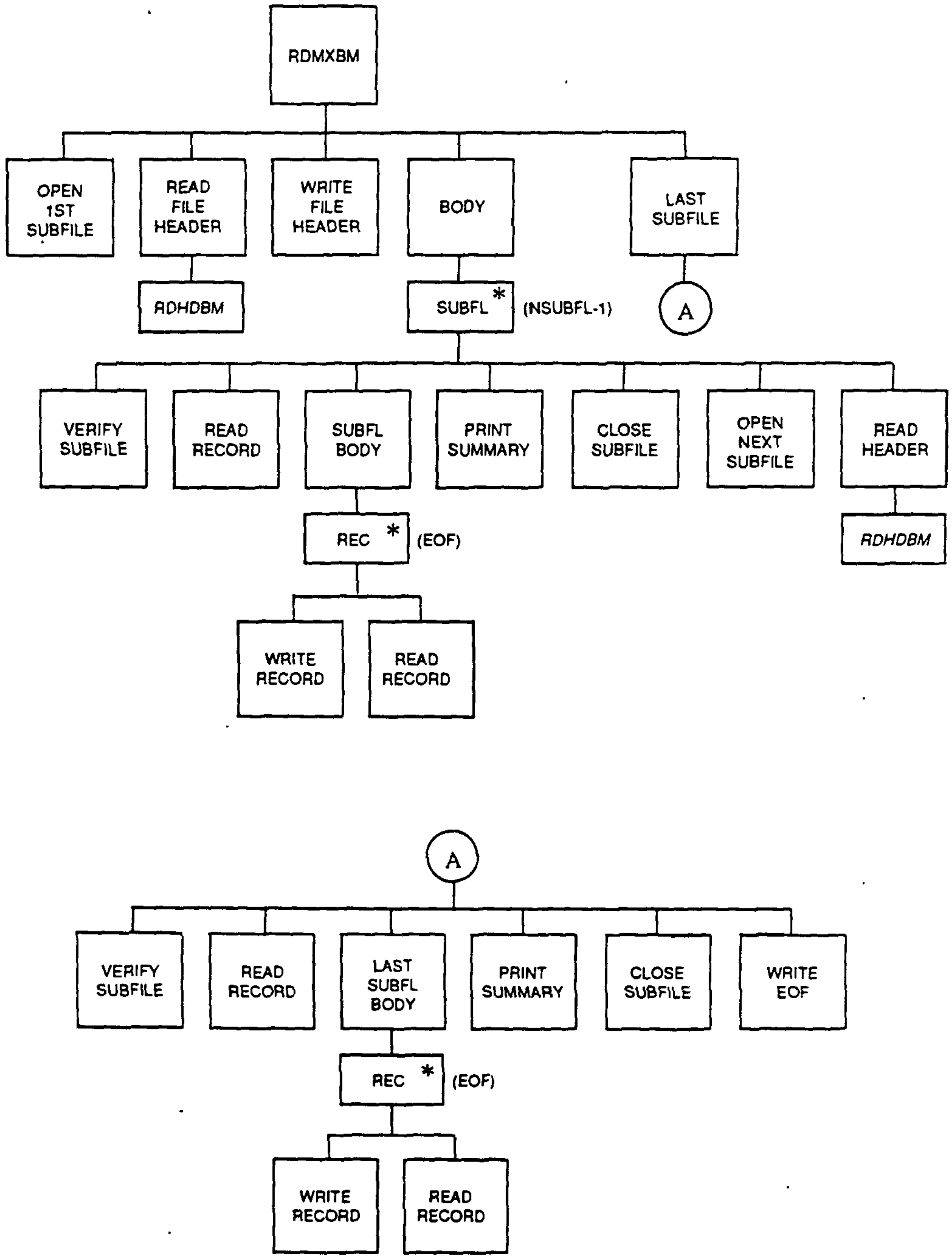


Figure C-15. RDMXBM (page 1 of 1).

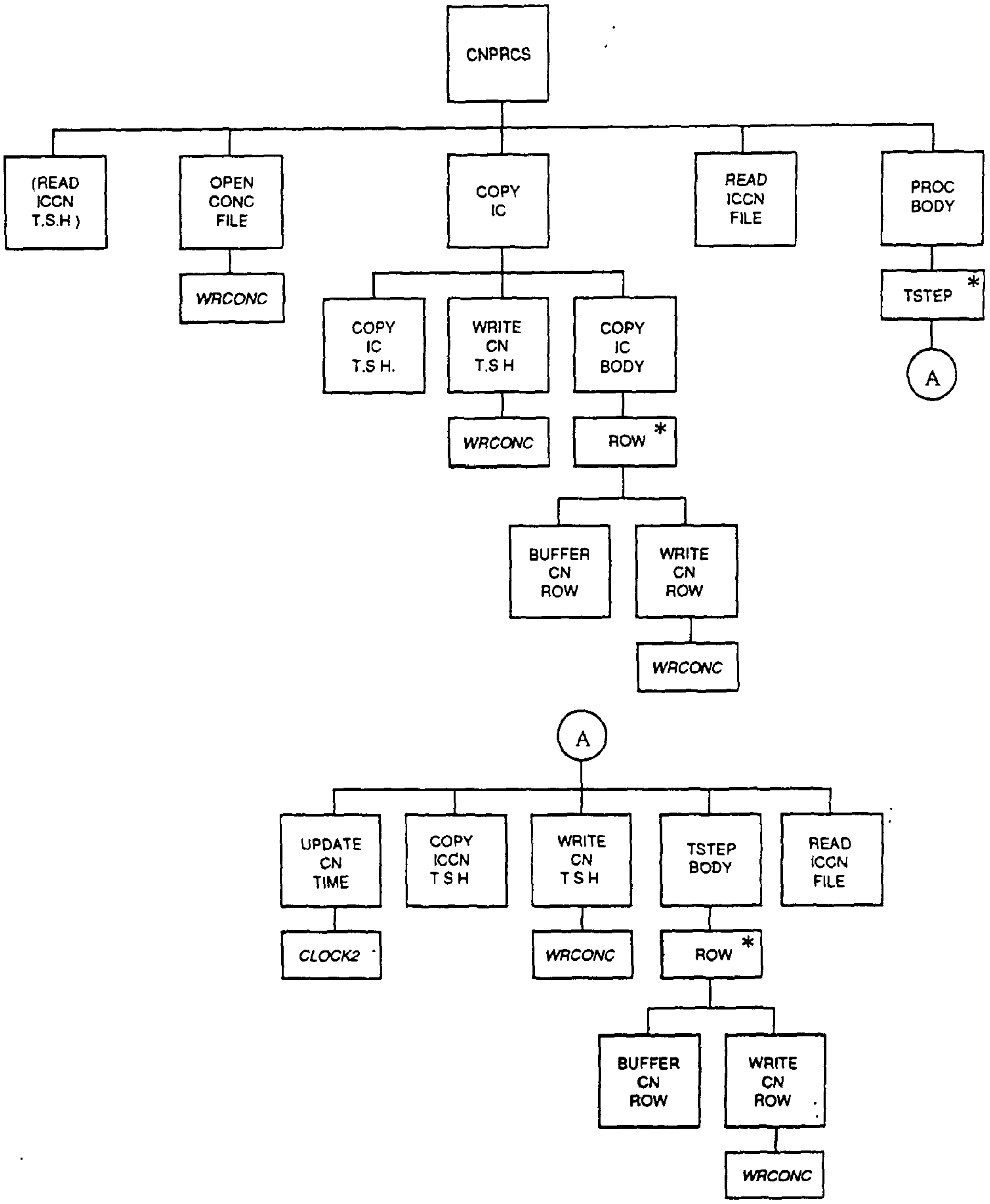


Figure C-16. CNPRCS (page 1 of 1).

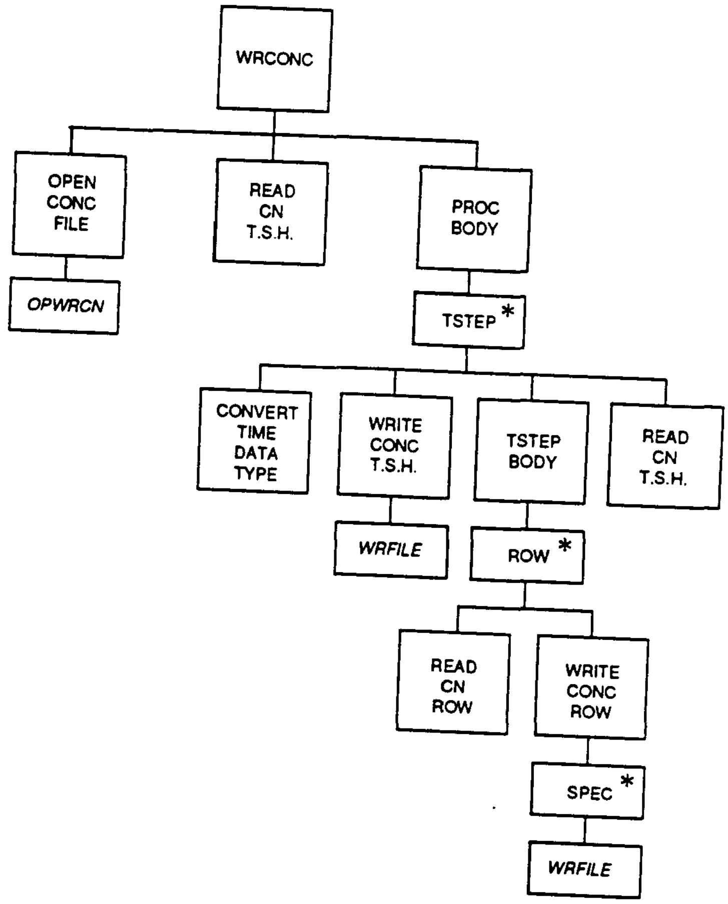


Figure C-17. WRCONC (page 1 of 1).

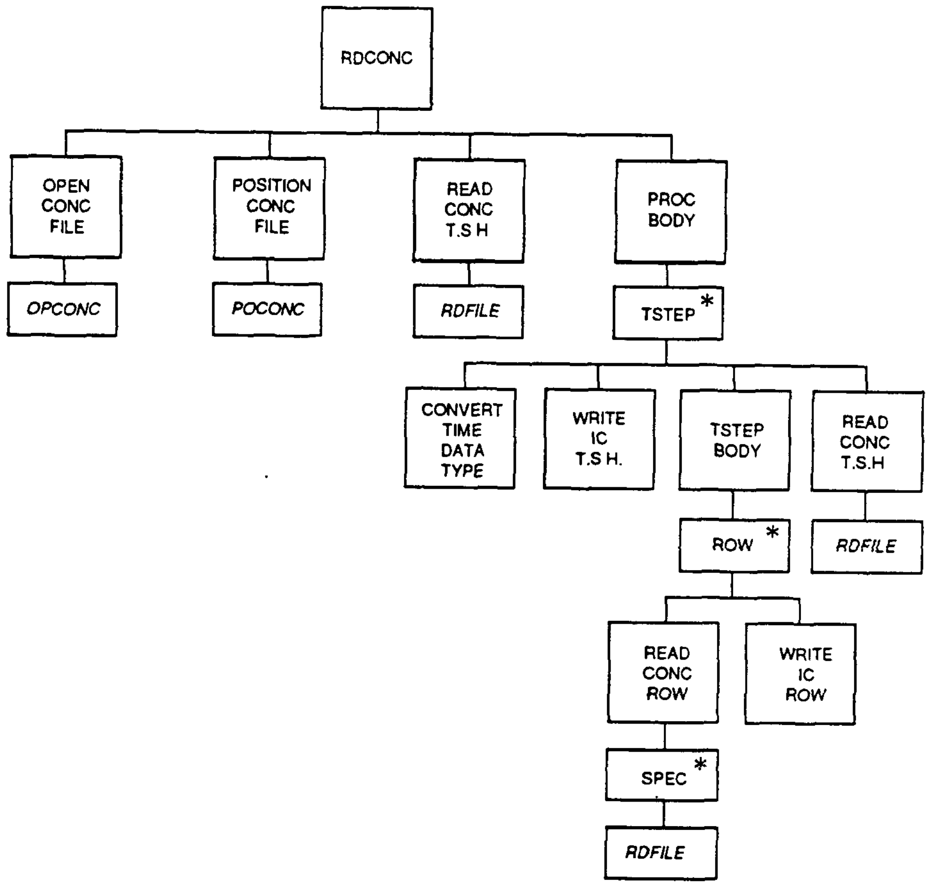


Figure C-18. RDCONC (page 1 of 1).

This page is intentionally left blank.

APPENDIX D

SAMPLE COMPILE AND LINK STREAM FOR THE UNIPROCESSOR ROM2.1

```
//GJDCROM  JOB (NER1RSMRP,B012),'JORDAN ',MSGCLASS=P,                   0000000
//         NOTIFY=GJD,TIME=(,45),PRTY=2                                 0000000
/*JOBPARM LINES=999
//*FTER JKVOFF03
/*ROUTE PRINT RMT378
//VSF2CL PROC FVPGM=FORTVS2,FVREGN=1400K,FVPDECK=NODECK,FVPOLST=NOLIST,
//          FVPOPT=0,FVTERM='SYSOUT=*',PGMNAME=ROM21R,PGMLIB='&&GOSET',
//            FVLNSPC='3200,(25,6)'
//FORT EXEC PGM=&FVPGM,REGION=&FVREGN,COND=(4,LT),
//          PARM='&FVPDECK,&FVPOLST,&OPT(&FVPOPT)'
//STEPLIB DD DSN=SYS1.VSF2COMP,DISP=SHR
//SYSPRINT DD SYSOUT=*,DCB=BLKSIZE=3429
//SYSTERM DD &FVTERM
//SYSPUNCH DD SYSOUT=B,DCB=BLKSIZE=3440
//SYSLIN DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//          SPACE=(&FVLNSPC),DCB=BLKSIZE=3200
//LKED EXEC PGM=IEWL,REGION=768K,COND=(4,LT),
//          PARM='LET,LIST,XREF'
//SYSPRINT DD SYSOUT=*
//SYSLIB DD DSN=SYS1.VSF2FORT,DISP=SHR
//       DD DSN=BTLNER1.ROM.LOADLIB,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(200,20))
//SYSLMOD DD DSN=&PGMLIB(&PGMNAME),DISP=(,PASS),UNIT=SYSDA,
//            SPACE=(TRK,(10,10,1),RLSE)
//SYSLIN DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//       DD DDNAME=SYSIN
//     PEND
//VSFVCL EXEC VSF2CL,PGMLIB='GJDNER1.ROMNET1.LOADLIB',                  00000300
// FVREGN=4M,FVLNSPC='3200,(250,60)',                                   00000400
// PARM.FORT='NODECK,NOLIST,OPT(2),CHARLEN(1024)',                      00000500
// PARM.LKED='LET,LIST,XREF'                                            00000600
//FORT.STEPLIB DD DSN=SYS1.VSF2COMP,DISP=SHR                            00000700
//FORT.SYSLIB DD DSN=GJDNER1.ROMNET1.INCLUDES,DISP=SHR                  00000800
//             DD DSN=GJDNER1.ROM1.INCLUDES,DISP=SHR                    00000800
//FORT.SYSIN DD DSN=GJDNER1.ROM1.FORT(ROM),DISP=SHR                     00000900
//LKED.SYSLMOD DD DSN=GJDNER1.ROMNET1.LOADLIB,DISP=SHR                  00001000
//LKED.SYSIN DD *                                                       00001100
     NAME ROM21R(R)                                                     00001200
/*
//                                                                      00000110
```

APPENDIX E

INCLUDE FILES

DIMENS.EXT and REGION.EXT contain the dimensions and definitions that are used by the ROM, and are tabulated first. The remaining include files are tabulated in alphabetical order.

TABLE E-1. INCLUDE FILE DIMENS.EXT

Referenced by	Description
BCPRCS, BIGGAM, BLKMOD, BMPRCS, BTPRCS, CNPRCS, DUMPHD, GTILDE, HSTEP, ICPRCS, INIRUN, LILGAM, NEWICS, OPBCON, OPBMAT, OPBTRK, OPCONC, OPICON, OPSTAV, OPWRCN, ORSPBC, ORSPBM, ORSPIC, POBCON, POBTRK, POCONC, POICON, POMXBM, POSTAV, PQ1, PQCOEF, RATED, RDBCON, RDBMAT, RDBT, RDBTRK, RDCONC, RDHDBM, RDICON, RDMXBM, RDSTAV, RTPHO, RTSET, RUNMGR, WRCONC, WRSTAV	Model dimensions

Variables	Type	Value	Description
NLEVS	Integer*4	3	Number of levels in model grid
			For Carbon Bond 4.2 Chemistry:
NSPECS	Integer*4	35	Number of chemical species in model
NRCT	Integer*4	84	Number of reactions
NPPRX	Integer*4	5	Number of primary photolytic reactions
NRKL	Integer*4	11	Number of levels at which photolytic rate constant is given
NPOXSP	Integer*4	3	Number of primary oxidant species (for dimensioning LGBMFL.EXT)

TABLE E-2. INCLUDE FILE REGION.EXT

Referenced by	Description
BCPRCS, BIGGAM, BMPRCS, BTPRCS, CNPRCS, ICPRCS, INIRUN, LILGAM, NEWICS, OPCONC, OPICON, OPSTAV, OPWRCN, PQCOEF, RDBCON, RDBMAT, RDBT, RDBTRK, RDCONC, RDICON, RDMXBM, RUNMGR, WRCONC, WRSTAV	Region-dependent parameters for the ROMNET domain

Variables	Type	Value	Description
GRDNAM	Character*8	'ROMNET'	Name of the region
NROWS	Integer*4	52	Number of rows in the model grid
NCOLS	Integer*4	64	Number of columns in the model grid

TABLE E-3. INCLUDE FILE ADVSFL.EXT

	Common block	Referenced by	Description
	ADVSFL	BIGGAM, LILGAM	This is the ADVECTION solution file. An advected row of BIGGAM is passed to LILGAM.
Variables	Type	Dimension	Description
ADVSFL(l,i,k)	Real*4	(NLEVS, NCOLS, NSPECS)	ADVECTION solution for one row

TABLE E-4. INCLUDE FILE BCFILE.EXT

	Common block	Referenced by	Description
	BCFILE	BCPRCS, RDBCON	The boundary values for each time step for species concentrations (by layer) for all modeling domain border cells.
Variables	Type	Dimension	Description
NORTH(i,l,k)	Real*4	(NCOLS, NLEVS, NSPECS)	Boundary condition concentrations along northern edge
WEST(j,l,k)	Real*4	(NROWS, NLEVS, NSPECS)	Boundary condition concentrations along western edge
EAST(j,l,k)	Real*4	(NROWS, NLEVS, NSPECS)	Boundary condition concentrations along eastern edge
SOUTH(i,l,k)	Real*4	(NCOLS, NLEVS, NSPECS)	Boundary condition concentrations along southern edge

TABLE E-5. INCLUDE FILE BGBCFL.EXT

	Common block	Referenced by	Description
	BGBCFL	BCPRCS, BIGGAM	Boundary conditions produced by BCPRCS for use by BIGGAM
Variables	Type	Dimension	Description
BCW(j,l,k)	Real*4	(NROWS, NLEVS, NSPECS)	Boundary condition concentrations along western edge
BCE(j,l,k)	Real*4	(NROWS, NLEVS, NSPECS)	Boundary condition concentrations along eastern edge
BCN(i,l,k)	Real*4	(NCOLS, NLEVS, NSPECS)	Boundary condition concentrations along northern edge
BCS(i,l,k)	Real*4	(NCOLS, NLEVS, NSPECS)	Boundary condition concentrations along southern edge

TABLE E-6. INCLUDE FILE BGBTFL.EXT

	Common block	Referenced by	Description
	BGBTFL	BIGGAM, BTPRCS	BTRK data for BIGGAM produced by BTPRCS
Variables	Type	Dimension	Description
			Backtrack locations:
XU(l,i,j)	Real*4	(NLEVS, NCOLS, NROWS)	Longitudinal component
XV(l,i,j)	Real*4	(NLEVS, NCOLS, NROWS)	Latitudinal component
			Horizontal diffusivities:
XKU(l,i,j)	Real*4	(NLEVS, NCOLS, NROWS)	Longitudinal component
XKV(l,i,j)	Real*4	(NLEVS, NCOLS, NROWS)	Latitudinal component

TABLE E-7. INCLUDE FILE BGICCN.EXT

	Common block	Referenced by	Description
	BGICCN	BIGGAM, CNPRCS, ICPRCS, LILGAM, NEWICS, RUNMGR	IC CONC file: time step initial concentrations for BIGGAM and time step concentrations finalized in LILGAM
Variables	Type	Dimension	Description
BGICCN(i,l,k,j,2)	Real*4	(NCOLS, NLEVS, NSPECS, NROWS, 2)	Time step initial concentrations for BIGGAM and time step concentrations finalized in LILGAM
TOG1	Integer*4		Pointer to upper half of BGICCN
TOG2	Integer*4		Pointer to lower half of BGICCN

TABLE E-8. INCLUDE FILE BMCOEF.EXT

	Common block	Referenced by	Description
	BMCOEF	HSTEP, LILGAM	BMAT coefficients from LILGAM for use by HSTEP
Variables	Type	Dimension	Description
			B-matrix coefficient in column *i* for species *k* for:
BB11(k)	Real*4	(NSPECS)	layer 1/surface 1 flux
BB22(k)	Real*4	(NSPECS)	layer 2/surface 2 flux
BB31(k)	Real*4	(NSPECS)	layer 3/surface 1 flux
			B-matrix coefficient in column *i* for:
BB12	Real*4		layer 1/surface 2 flux
BB21	Real*4		layer 2/surface 1 flux
BB23	Real*4		layer 2/surface 3 flux
BB32	Real*4		layer 3/surface 2 flux
BB33	Real*4		layer 3/surface 3 flux
			B-matrix coefficient for subgrid scale adjustment in column *i* for:
BB11S	Real*4		layer 1/surface 1 flux
BB11SS	Real*4		alternate layer 1/surface 1 flux
BB31S	Real*4		layer 3/surface 1 flux
BB31SS	Real*4		alternate layer 3/surface 1 flux

TABLE E-9. INCLUDE FILE BMFILE.EXT

	Common block	Referenced by	Description
	BMFILE	BMPRCS, RDBMAT	Data from BMAT read by RDBMAT to be passed to BMPRCS
Variables	Type	Dimension	Description
			B-matrix coefficient in column *i* for:
XB12(i)	Real*4	(NCOLS)	layer 1/surface 2 flux
XB13(i)	Real*4	(NCOLS)	layer 1/surface 3 flux
XB21(i)	Real*4	(NCOLS)	layer 2/surface 1 flux
XB23(i)	Real*4	(NCOLS)	layer 2/surface 3 flux
XB32(i)	Real*4	(NCOLS)	layer 3/surface 2 flux
XB33(i)	Real*4	(NCOLS)	layer 3/surface 3 flux
			B-matrix coefficient for subgrid scale adjustment in column *i* for:
XB11S(i)	Real*4	(NCOLS)	layer 1/surface 1 flux
XB11SS(i)	Real*4	(NCOLS)	alternate layer 1/surface 1 flux
XB31S(i)	Real*4	(NCOLS)	layer 3/surface 1 flux
XB31SS(i)	Real*4	(NCOLS)	alternate layer 3/surface 1 flux
			Run time subgrid scale adjustment parameters in column i:
XQO3FC(i)	Real*4	(NCOLS)	ozone factor
XSS0NO(i)	Real*4	(NCOLS)	NO surface emissions source strength
XG1S(i,k)	Real*4	(NCOLS, NPOXSP)	emissions source term in layer 1 for primary oxidant species *k*
XG1SS(i,k)	Real*4	(NCOLS, NPOXSP)	alternate emissions source term in layer 1 for primary oxidant species *k*
XG1FAC(i,k)	Real*4	(NCOLS, NPOXSP)	emissions source factor in layer 1 for primary oxidant species *k*
XG3S(i,k)	Real*4	(NCOLS, NPOXSP)	emissions source term in layer 3 for primary oxidant species *k*
XG3SS(i,k)	Real*4	(NCOLS, NPOXSP)	alternate emissions source term in layer 3 for primary oxidant species *k*
XG3FAC(i,k)	Real*4	(NCOLS, NPOXSP)	emissions source factor in layer 3 for primary oxidant species *k*
XRHO(i,l)	Real*4	(NCOLS, NLEVS)	Rate constants density correction factor in column *i* for layer *l*
XTEMP(i,l)	Real*4	(NCOLS, NLEVS)	Absolute temperature for rate constants adjustment in column *i* for layer *l*
XWVC(i,l)	Real*4	(NCOLS, NLEVS)	Water vapor concentration for rate constants adjustment in column *i* for layer *l*
XTHETA(i)	Real*4	(NCOLS)	Solar zenith angle for photolytic rate constants adjustment in column *i*
XPSI2(i)	Real*4	(NCOLS)	Cloud cover correction factor for photolytic rate constants adjustment in column *i*
			Heights above sea level in column *i* (used for rate constant adjustments):
XZ0(i)	Real*4	(NCOLS)	Layer 0
XZ1(i)	Real*4	(NCOLS)	Layer 1
XZ2(i)	Real*4	(NCOLS)	Layer 2
XZ3(i)	Real*4	(NCOLS)	Layer 3
			Buffer containing six species-dependent B-matrix variables:
AA1(i,6,k)	Real*4	(NCOLS, 6, NSPECS)	B11, B22, B31, G1, G2, G3

TABLE E-10. INCLUDE FILE BTFILE.EXT

Variables	Common block	Referenced by	Description
	BTFILE	BTPRCS, RDBTRK	BTRK file block data from the BTRK file read by RDBTRK to be passed to BTPRCS
Variables	**Type**	**Dimension**	**Description**
			Backtrack locations:
XRU(i,l)	Real*4	(NCOLS, NLEVS)	Longitudinal component, column i, layer *l*
XRV(i,l)	Real*4	(NCOLS, NLEVS)	Latitudinal component, column i, layer *l*
			Horizontal Diffusivities:
XRKU(i,l)	Real*4	(NCOLS, NLEVS)	Longitudinal component, column i, layer *l*
XRKV(i,l)	Real*4	(NCOLS, NLEVS)	Latitudinal component, column i, layer *l*

TABLE E-11. INCLUDE FILE CHEMIN.EXT

Variables	Common block	Referenced by	Description
	CHEMIN	INIRUN, GTILDE, HSTEP, LILGAM, OPSTAV, WRSTAV	Input control for LILGAM chemistry calculations
Variables	**Type**	**Dimension**	**Description**
ATS	Real*4		ADVECTION time step length (seconds)
GTS	Real*4		VERTICAL FLUX time step length (seconds)
UFRAX	Real*4		Upper tolerance for FRAX (FRAX = time step tolerance for change in species value to set the chemistry time step)
BFRAX	Real*4		Lower tolerance for FRAX
FACTOR	Real*4		Fraction of ($NO + NO_2 + O_3$) to represent lower limit of species concentration considered in time step computations
DIVP	Real*4		Multiplier (0 .LT. DIVP .LE. 1.0) to determine intermediate value of P1 to be used in CORRECTOR chemistry solution per time step
DIVQ	Real*4		Multiplier (0 .LT. DIVQ .LE. 1.0) to determine intermediate value of Q1 to be used in CORRECTOR chemistry solution per time step
NCOUT	Integer*4		Number of species (3) whose concentrations are used to determine which species are included in chemistry time step computations
ISPEC(k)	Integer*4	(NSPECS)	List of species indices whose concentrations are used to determine which species are included in chemistry time step computations
ULIM	Real*4		Upper limit of chemistry time step length (seconds)
BLIM	Real*4		Lower limit of chemistry time step length (seconds)
FNOLIM	Real*4		Upper limit to control UFRAX/BFRAX chemistry computation accuracy

TABLE E-12. INCLUDE FILE CHEMSW.EXT

	Common block	Referenced by	Description
	CHEMSW	LILGAM, RUNMGR	Chemistry on/off flag
Variables	Type	Dimension	Description
CHEMON	Logical*4		= .TRUE. ⇒ LILGAM normal run = .FALSE. ⇒ LILGAM chemistry calculations bypassed

TABLE E-13. INCLUDE FILE CNFILE.EXT

	Common block	Referenced by	Description
	CNFILE	CNPRCS, RDCONC, RUNMGR, WRCONC	Final solution concentrations passed to CNPRCS by LILGAM
Variables	Type	Dimension	Description
CNFILE(i,l,k)	Real*4	(NCOLS, NLEVS, NSPECS)	Solution concentrations for one row

TABLE E-14. INCLUDE FILE CONFAC.EXT

	Common block	Referenced by	Description
	CONFAC	BIGGAM, OPSTAV, WRSTAV	Diffusivities conversion factors for BIGGAM
Variables	Type	Dimension	Description
RDLNT2	Real*4		Longitudinal horizontal diffusivities conversion factor
RDLTT2	Real*4		Latitudinal horizontal diffusivities conversion factor

TABLE E-15. INCLUDE FILE ERRG.EXT

	Common block	Referenced by	Description
	ERRGT	GTILDE, LILGAM	GTILDE error counts
Variables	Type	Dimension	Description
NGTILN(k)	Integer*4	(NSPECS)	Count of number of negative GTILDE's computed

TABLE E-16. INCLUDE FILE FLNAMS.EXT

	Common block	Referenced by	Description
	FLNAMS	BLKMOD, DUMPHD, NEWICS, OPBCON, OPCONC, OPICON, OPSTAV, OPWRCN, RDBT, RDMXBM, RUNMGR, WRSTAV	Internal program names used for file operating system
Variables	Type	Dimension	Description
FLNMBC	Character*12		Internal name of BCON file
FLNMBM(m)	Character*12	(NUMBMF)	Internal name of BMAT subfile *m*
FLNMBT	Character*12		Internal name of BTRK file
FLNMCN	Character*12		Internal name of CONC file
FLNMIC	Character*12		Internal name of ICON file
FLNMSV	Character*12		Internal name of STATE VECTOR file
FLNMNI	Character*12		Internal name of NEW IC file
FLNMST	Character*12		Internal name of STOP file
FLNMPR	Character*12		Internal name of PROGRESS file
NUMBMF	Integer*4		Maximum number of multiple BMAT subfiles (=6; passed from common block LUNITS)

TABLE E-17. INCLUDE FILE GTCOEF.EXT

	Common block	Referenced by	Description
	COEFS	GTILDE, LILGAM	Terms in 3 × 3 system of differential equations solved by GTILDE
Variables	Type	Dimension	Description
X0	Real*4		Initial condition (time t_0) for GTILDE
Y0	Real*4		Initial condition (time t_0) for GTILDE
Z0	Real*4		Initial condition (time t_0) for GTILDE
A1	Real*4		Coefficient in differential equations
A2	Real*4		Coefficient in differential equations
A3	Real*4		Coefficient in differential equations
B1	Real*4		Coefficient in differential equations
B2	Real*4		Coefficient in differential equations
B3	Real*4		Coefficient in differential equations
C2	Real*4		Coefficient in differential equations
C3	Real*4		Coefficient in differential equations
GS1	Real*4		Inhomogeneous source term in differential equations
GS2	Real*4		Inhomogeneous source term in differential equations
GS3	Real*4		Inhomogeneous source term in differential equations

TABLE E-18. INCLUDE FILE HDFMTS.EXT

	Common block	Referenced by	Description
	HDFMTS	BLKMOD, DUMPHD, NEWICS, OPBCON, OPBMAT, OPBTRK, OPCONC, OPICON, OPSTAV, OPWRCN, WRSTAV	Formats for headers
Variables	Type	Dimension	Description
			Buffers that contain Fortran formats for echoing file header information to the log:
HDALL	Character*488		First segment record log format
HSPN	Character*48		Species record log format
HLVN	Character*48		Level record log format
HTEXT	Character*48		Text record log format

TABLE E-19. INCLUDE FILE HDSTAV.EXT

Common block	Referenced by	Description
CHARSV, HEADSV	OPSTAV, POSTAV, RUNMGR, WRSTAV	Formatted STATE VECTOR file header block

CHARSV

Variables	Type	Dimension	Description
GRDNSV	Character*8		Grid definition name
SPNMSV(k)	Character*4	(NSPECS)	Name of species *k*
LVNMSV(l)	Character*4	(NLEVS)	Name of level *l*
TEXTSV(20)	Character*80	(20)	Text description

HEADSV

Variables	Type	Dimension	Description
CDATSV	Integer*4		Creation date (*MMDDYY*) of STATE VECTOR file
CTIMSV	Integer*4		Creation time (*HHMMSS*) of STATE VECTOR file
SDATSV	Integer*4		Start date (*YYDDD*) of scenario
STHRSV	Integer*4		Start hour (00 to 23) of scenario
TSTPSV	Integer*4		Size of step (seconds) used in simulation
FRSTSV	Integer*4		First time step (seconds past start time of model scenario) in simulation
SWLNSV	Real*4		Longitude (degrees) of southwest corner of grid
SWLTSV	Real*4		Latitude (degrees) of southwest corner of grid
NELNSV	Real*4		Longitude (degrees) of northeast corner of grid
NELTSV	Real*4		Latitude (degrees) of northeast corner of grid
DLONSV	Real*4		Longitudinal increment cell (degrees)
DLATSV	Real*4		Latitudinal cell increment (degrees)
NCOLSV	Integer*4		Number of columns in grid (= NCOLS)
NROWSV	Integer*4		Number of rows in grid (= NROWS)
NLEVSV	Integer*4		Number of levels (= NLEVS)
NSPCSV	Integer*4		Number of species in simulation (= NSPECS)
ICNTSV	Integer*4		Number of text records in header

TABLE E-20. INCLUDE FILE HEADBC.EXT

Common block	Referenced by	Description
CHARBC, HEADBC	BCPRCS, OPBCON, OPWRCN, ORSPBC, POBCON, RDBCON, RUNMGR	BCON file header block

CHARBC

Variables	Type	Dimension	Description
GRDNBC	Character*8		Grid definition name
SPNMBC(k)	Character*4	(NSPECS)	Name of species *k*
LVNMBC(l)	Character*4	(NLEVS)	Name of level *l*
TEXTBC(20)	Character*80	(20)	Text description

HEADBC

Variables	Type	Dimension	Description
CDATBC	Integer*4		Creation date (*MMDDYY*) of BC file
CTIMBC	Integer*4		Creation time (*HHMMSS*) of BC file
SDATBC	Integer*4		Start date (*YYDDD*) of scenario on BC file
STHRBC	Integer*4		Start hour (00 to 23) of scenario on BC file
TSTPBC	Integer*4		Size of time step (seconds) used in BC file
FRSTBC	Integer*4		First time step (seconds past start time of BCON scenario) on BCON file
SWLNBC	Real*4		Longitude (degrees) of southwest corner of grid
SWLTBC	Real*4		Latitude (degrees) of southwest corner of grid
NELNBC	Real*4		Longitude (degrees) of northeast corner of grid
NELTBC	Real*4		Latitude (degrees) of northeast corner of grid
DLONBC	Real*4		Longitudinal increment cell (degrees)
DLATBC	Real*4		Latitudinal cell increment (degrees)
NCOLBC	Integer*4		Number of columns in grid
NROWBC	Integer*4		Number of rows in grid
NLEVBC	Integer*4		Number of levels
NSPCBC	Integer*4		Number of species in BCON file
ICNTBC	Integer*4		Number of text records in header

TABLE E-21. INCLUDE FILE HEADBM.EXT

Common block	Referenced by	Description
CHARBM, HEADBM, BMSPRD	BMPRCS, DUMPHD, OPBMAT, OPWRCN, ORSPBM, POMXBM, RDBMAT, RDHDBM, RDMXBM, RUNMGR	BMAT file header block

Variables	Type	Value	Description
NMFBM	Integer*4	4	Number of MIF files used in generating BMAT file
NVRBM1	Integer*4	18+3xNLEVS+6xNPOXSP	Number of part 1 BMAT variables, specifically: B12, B13, B21, B23, B32, B33, B11S, B11SS, B31S, B31SS, QO3FAC, SSONO, TTHETA, PPSI2, ZZ0, ZZ1, ZZ2, ZZ3; RRHO(NLEVS)'s, TTEMP(NLEVS)'s, WWVC(NLEVS)'s; and G1S, G1SS, G1FAC, G3S, G3SS, G3FAC, each dimensioned by NPOXSP
NVRBM2	Integer*4	6	Number of species array BMAT variables (part 2), specifically: B11, B22, B31, G1, G2, G3

CHARBM

Variables	Type	Dimension	Description
GRDNBM	Character*8		Grid definition name
SPNMBM(k)	Character*4	(NSPECS)	Name of species *k*
LVNMBM(l)	Character*4	(NLEVS)	Name of level *l*
TEXTBM(20)	Character*80	(20)	Text description
MFNMBM(m)	Character*12	(NMFBM)	File name of MIF file *m* used to produce the BMAT file

HEADBM

Variables	Type	Dimension	Description
ISUBFL	Integer*4		Ordinal subfile number
NSUBFL	Integer*4		Total number of subfiles of BMAT file
FRSTSF	Integer*4		First time step on subfile ISUBFL
LSSTSF	Integer*4		Last time step on subfile ISUBFL
CDATBM	Integer*4		Creation date (*MMDDYY*) of BMAT file
CTIMBM	Integer*4		Creation time (*HHMMSS*) of BMAT file
SDATBM	Integer*4		Start date (*YYDDD*) of scenario
STHRBM	Integer*4		Start hour (00 to 23) of scenario
TSTPBM	Integer*4		Time step size (seconds) for simulation
FRSTBM	Integer*4		First time step (seconds past start time of scenario) on BMAT file
SWLNBM	Real*4		Longitude (degrees) of southwest corner of grid
SWLTBM	Real*4		Latitude (degrees) of southwest corner of grid
NELNBM	Real*4		Longitude (degrees) of northeast corner of grid
NELTBM	Real*4		Latitude (degrees) of northeast corner of grid
DLONBM	Real*4		Longitudinal cell increment (degrees)
DLATBM	Real*4		Latitudinal cell increment (degrees)
NCOLBM	Integer*4		Number of columns in grid
NROWBM	Integer*4		Number of rows in grid
NLEVBM	Integer*4		Number of levels
NSPCBM	Integer*4		Number of species on BMAT file
NMIFBM	Integer*4		Number of MIF files used to generate BMAT
CDMFBM(m)	Integer*4	(NMFBM)	Creation date of MIF file *m* used to produce the BMAT file
CTMFBM(m)	Integer*4	(NMFBM)	Creation date of MIF file *m* used to produce the BMAT file
UDMFBM(m)	Integer*4	(NMFBM)	Last update date of MIF file *m* used to produce the BMAT file
UTMFBM(m)	Integer*4	(NMFBM)	Last update time of MIF file *m* used to produce the BMAT file
ICNTBM	Integer*4		Number of text records on header
BMINDX(k)	Integer*4	(NSPECS)	Index array for species expansion from reduced BMAT set to full MODEL set

BMSPRD

Variables	Type	Dimension	Description
BMSPRD	Integer*4		Number of species in BMAT reduced set representing NSPCBM species.

TABLE E-22. INCLUDE FILE HEADBT.EXT

Common block	Referenced by	Description
CHARBT, HEADBT	BTPRCS, OPBTRK, OPWRCN, POBTRK, RDBT, RDBTRK, RUNMGR	BTRK file header block

Variables	Type	Value	Description
NMFBT	Integer*4	6	Number of MIF files used in generating BTRK file
NVARBT	Integer*4	4	Number of LEVEL-DEPENDENT BTRK variables, specifically: UU's, VV's, KKHU's, KKHV's

CHARBT

Variables	Type	Dimension	Description
GRDNBT	Character*8		Grid definition name
TEXTBT(20)	Character*80	(20)	Text description
MFNMBT(m)	Character*12	(NMFBT)	File name of MIF file *m* used to produce the BTRK file

HEADBT

Variables	Type	Dimension	Description
CDATBT	Integer*4		Creation date (*MMDDYY*) of BTRK file
CTIMBT	Integer*4		Creation time (*HHMMSS*) of BTRK file
SDATBT	Integer*4		Start date (*YYDDD*) of scenario
STHRBT	Integer*4		Start hour (00 to 23) of scenario
TSTPBT	Integer*4		Time step size (seconds) for simulation
FRSTBT	Integer*4		First time step (seconds past start time of scenario) on BTRK file
SWLNBT	Real*4		Longitude (degrees) of southwest corner of grid
SWLTBT	Real*4		Latitude (degrees) of southwest corner of grid
NELNBT	Real*4		Longitude (degrees) of northeast corner of grid
NELTBT	Real*4		Latitude (degrees) of northeast corner of grid
DLONBT	Real*4		Longitudinal cell increment (degrees)
DLATBT	Real*4		Latitudinal cell increment (degrees)
NCOLBT	Integer*4		Number of columns in grid
NROWBT	Integer*4		Number of rows in grid
NMIFBT	Integer*4		Number of MIF files used to generate BTRK file
CDMFBT(m)	Integer*4	(NMFBT)	Creation date of MIF file *m* used to produce the BTRK file
CTMFBT(m)	Integer*4	(NMFBT)	Creation date of MIF file *m* used to produce the BTRK file
UDMFBT(m)	Integer*4	(NMFBT)	Last update date of MIF file *m* used to produce the BTRK file
UTMFBT(m)	Integer*4	(NMFBT)	Last update time of MIF file *m* used to produce the BTRK file
ICNTBT	Integer*4		Number of text records in header

TABLE E-23. INCLUDE FILE HEADCN.EXT

Common block	Referenced by	Description
CHARCN, HEADCN	CNPRCS, NEWICS, OPCONC, OPWRCN, POCONC, RDCONC, RUNMGR	CONC file header block

CHARCN

Variables	Type	Dimension	Description
GRDNCN	Character*8		Grid definition name
SPNMCN(k)	Character*4	(NSPECS)	Name of species *k*
LVNMCN(l)	Character*4	(NLEVS)	Name of level *l*
TEXTCN(20)	Character*80	(20)	Text description

HEADCN

Variables	Type	Dimension	Description
Variables	Type	Dimension	Description
CDATCN	Integer*4		Creation date (*MMDDYY*) of CONC file
CTIMCN	Integer*4		Creation time (*HHMMSS*) of CONC file
SDATCN	Integer*4		Start date (*YYDDD*) of scenario
STHRCN	Integer*4		Start hour (00 to 23) of scenario
TSTPCN	Integer*4		Size of step (seconds) used in simulation
FRSTCN	Integer*4		First time step (seconds past start time of model scenario) on CONC file
SWLNCN	Real*4		Longitude (degrees) of southwest corner of grid
SWLTCN	Real*4		Latitude (degrees) of southwest corner of grid
NELNCN	Real*4		Longitude (degrees) of northeast corner of grid
NELTCN	Real*4		Latitude (degrees) of northeast corner of grid
DLONCN	Real*4		Longitudinal increment cell (degrees)
DLATCN	Real*4		Latitudinal cell increment (degrees)
NCOLCN	Integer*4		Number of columns in grid (= NCOLS)
NROWCN	Integer*4		Number of rows in grid (= NROWS)
NLEVCN	Integer*4		Number of levels (= NLEVS)
NSPCCN	Integer*4		Number species in CONC file (= NSPECS)
ICNTCN	Integer*4		Number of text records in header
CDBMCN	Integer*4		Creation date (*MMDDYY*) of BMAT file used in simulation
CTBMCN	Integer*4		Creation time (*HHMMSS*) of BMAT file used in simulation
CDBTCN	Integer*4		Creation date (*MMDDYY*) of BTRK file used in simulation
CTBTCN	Integer*4		Creation time (*HHMMSS*) of BTRK file used in simulation
CDBCCN	Integer*4		Creation date (*MMDDYY*) of BCON file used in simulation
CTBCCN	Integer*4		Creation time (*HHMMSS*) of BCON file used in simulation
CDICCN	Integer*4		Creation date (*MMDDYY*) of ICON file used in simulation
CTICCN	Integer*4		Creation time (*HHMMSS*) of ICON file used in simulation

TABLE E-24. INCLUDE FILE HEADIC.EXT

Common block	Referenced by	Description
CHARIC, HEADIC	ICPRCS, NEWICS, OPICON, OPWRCN, ORSPIC, POICON, RDICON, RUNMGR	ICON file header block

CHARIC

Variables	Type	Dimension	Description
GRDNIC	Character*8		Grid definition name
SPNMIC(k)	Character*4	(NSPECS)	Name of species *k*
LVNMIC(l)	Character*4	(NLEVS)	Name of level *l*
TEXTIC(20)	Character*80	(20)	Text description

HEADIC

Variables	Type	Dimension	Description
CDATIC	Integer*4		Creation date (*MMDDYY*) of ICON file
CTIMIC	Integer*4		Creation time (*HHMMSS*) of ICON file
SDATIC	Integer*4		Start date (*YYDDD*) of scenario
STHRIC	Integer*4		Start hour (00 to 23) of scenario
TSTPIC	Integer*4		Size of step (seconds) used in simulation
FRSTIC	Integer*4		First time step (seconds past start time of model scenario) on ICON file
SWLNIC	Real*4		Longitude (degrees) of southwest corner of grid
SWLTIC	Real*4		Latitude (degrees) of southwest corner of grid
NELNIC	Real*4		Longitude (degrees) of northeast corner of grid
NELTIC	Real*4		Latitude (degrees) of northeast corner of grid
DLONIC	Real*4		Longitudinal increment cell (degrees)
DLATIC	Real*4		Latitudinal cell increment (degrees)
NCOLIC	Integer*4		Number of columns in GRID (= NCOLS)
NROWIC	Integer*4		Number of rows in GRID (= NROWS)
NLEVIC	Integer*4		Number of levels (= NLEVS)
NSPCIC	Integer*4		Number of species in ICON file (= NSPECS)
IDUM(8)	Integer*4	(8)	Padding
ICNTIC	Integer*4		Number of text records

TABLE E-25. INCLUDE FILE HEADIN.EXT

Common block	Referenced by	Description
CHARIN, HEADIN	BCPRCS, BIGGAM, BLKMOD, BMPRCS, BTPRCS, CNPRCS, DUMPHD, HSTEP, ICPRCS, INIRUN, LILGAM, NEWICS, OPSTAV, OPWRCN, ORSPBC, ORSPBM, ORSPIC, PQ1, RDBCON, RDBMAT, RDBTRK, RDCONC, RDICON, RUNMGR, WRSTAV	Input header information, used to check file headers on BMAT, ICON, and BCON files, and to create file header on CONC file.

CHARIN

Variables	Type	Dimension	Description
GRDNIN	Character*8		Grid definition name
SPNMIN(k)	Character*4	(NSPECS)	Name of species *k*
LVNMIN(l)	Character*4	(NLEVS)	Name of level *l*
TEXTIN(20)	Character*80	(20)	Text description

HEADIN

Variables	Type	Dimension	Description
CDATIN	Integer*4		Creation date of simulation (YYMMDD)
CTIMIN	Integer*4		Creation time of simulation (*HHMMSS*)
SDATIN	Integer*4		5-digit JULIAN start date (*YYDDD*) for model scenario
STHRIN	Integer*4		Start hour (00 TO 23) for model scenario
TSTPIN	Integer*4		Step size (seconds) for model scenario
FRSTIN	Integer*4		First time step (seconds past start time of model scenario) in simulation
SWLNIN	Real*4		Longitude (degrees) of southwest corner of grid
SWLTIN	Real*4		Latitude (degrees) of southwest corner of grid
NELNIN	Real*4		Longitude (degrees) of northeast corner of grid
NELTIN	Real*4		Latitude (degrees) of northeast corner of grid
DLONIN	Real*4		Longitudinal increment cell (degrees)
DLATIN	Real*4		Latitudinal cell increment (degrees)
NCOLIN	Integer*4		Number of columns in grid (= NCOLS)
NROWIN	Integer*4		Number of rows in grid (= NROWS)
NLEVIN	Integer*4		Number of levels (= NLEVS)
NSPCIN	Integer*4		Number of species in model
ICNTIN	Integer*4		Number of text records in header

TABLE E-26. INCLUDE FILE HSTEPS.EXT

	Common block	Referenced by	Description
	HSTEPS	HSTEP, LILGAM	Integration steps (H) for GTILDE computed by HSTEP
Variables	Type	Dimension	Description
HSTEPS(k)	Real*4	(NSPECS)	Integration step
HSTPO3(2)	Real*4	(2)	Integration step for OZONE, two possible conditions
IND(k)	Integer*4	(NSPECS)	Indicator of success in root calculation from HSTEP = 1 ⇒ valid root = 2 ⇒ invalid root
INDO3(2)	Integer*4	(2)	Indicator of success in root calculation from HSTEP

TABLE E-27. INCLUDE FILE ICFILE.EXT

	Common block	Referenced by	Description
	ICFILE	ICPRCS, NEWICS, RDICON	Initial conditions file read by ICPRCS
Variables	Type	Dimension	Description
ICFILE(i,l,k)	Real*4	(NCOLS, NLEVS, NSPECS)	Initial concentrations for one row

TABLE E-28. INCLUDE FILE LGBMFL.EXT

	Common block	Referenced by	Description
	LGBMFL	BMPRCS, LILGAM	Intermediate BMAT file read by LILGAM written by BMPRCS
Variables	Type	Dimension	Description
			B-matrix coefficient in column *i* for:
B12(i)	Real*4	(NCOLS)	layer 1/surface 2 flux
B13(i)	Real*4	(NCOLS)	layer 1/surface 3 flux
B21(i)	Real*4	(NCOLS)	layer 2/surface 1 flux
B23(i)	Real*4	(NCOLS)	layer 2/surface 3 flux
B32(i)	Real*4	(NCOLS)	layer 3/surface 2 flux
B33(i)	Real*4	(NCOLS)	layer 3/surface 3 flux
			B-matrix coefficient in column *i* for species *k* for:
B11(i,k)	Real*4	(NCOLS, NSPECS)	layer 1/surface 1 flux
B22(i,k)	Real*4	(NCOLS, NSPECS)	layer 2/surface 2 flux
B31(i,k)	Real*4	(NCOLS, NSPECS)	layer 3/surface 1 flux
			Emissions source term in column *i* for species *k* for:
G1(i,k)	Real*4	(NCOLS, NSPECS)	layer 1
G2(i,k)	Real*4	(NCOLS, NSPECS)	layer 2
G3(i,k)	Real*4	(NCOLS, NSPECS)	layer 3
			B-matrix coefficient for subgrid scale adjustment in column *i* for:
B11S(i)	Real*4	(NCOLS)	layer 1/surface 1 flux
B11SS(i)	Real*4	(NCOLS)	alternate layer 1/surface 1 flux
B31S(i)	Real*4	(NCOLS)	layer 3/surface 1 flux
B31SS(i)	Real*4	(NCOLS)	alternate layer 3/surface 1 flux
			Run time subgrid scale adjustment parameters in column i:
QO3FAC(i)	Real*4	(NCOLS)	ozone factor
SS0NO(i)	Real*4	(NCOLS)	NO surface emissions source strength
G1S(i,k)	Real*4	(NCOLS, NPOXSP)	emissions source term in layer 1 for primary oxidant species *k*
G1SS(i,k)	Real*4	(NCOLS, NPOXSP)	alternate emissions source term in layer 1 for primary oxidant species *k*
G1FAC(i,k)	Real*4	(NCOLS, NPOXSP)	emissions source factor in layer 1 for primary oxidant species *k*
G3S(i,k)	Real*4	(NCOLS, NPOXSP)	emissions source term in layer 3 for primary oxidant species *k*
G3SS(i,k)	Real*4	(NCOLS, NPOXSP)	alternate emissions source term in layer 3 for primary oxidant species *k*
G3FAC(i,k)	Real*4	(NCOLS, NPOXSP)	emissions source factor in layer 3 for primary oxidant species *k*
RHO(i,l)	Real*4	(NCOLS, NLEVS)	Rate constants density correction factor in column *i* for layer *l*
TEMP(i,l)	Real*4	(NCOLS, NLEVS)	Absolute temperature for rate constants adjustment in column *i* for layer *l*
WVC(i,l)	Real*4	(NCOLS, NLEVS)	Water vapor concentration for rate constants adjustment in column *i* for layer *l*
THETA(i)	Real*4	(NCOLS)	Solar zenith angle for photolytic rate constants adjustment in column *i*
PSI2(i)	Real*4	(NCOLS)	Cloud cover correction factor for photolytic rate constants adjustment in column *i*
ZLEV(i,l)	Real*4	(NCOLS, NLEVS + 1)	Heights above sea level in column *i* (used for rate constant adjustments) for layer *l*

TABLE E-29. INCLUDE FILE LILGSP.EXT

	Common block	Referenced by	Description
	LILGSP	HSTEP, LILGAM, OPSTAV, PQ1, WRSTAV	Contains special SPECIES control for LILGAM
Variables	**Type**	**Dimension**	**Description**
INBIG3(k)	Logical*4	(NSPECS)	.TRUE. for SPECIES NO, NO_2, O_3, otherwise .FALSE.
NOHIT	Integer*4		Index for NO
NO2HIT	Integer*4		Index for NO_2
O3HIT	Integer*4		Index for O_3
OLEHIT	Integer*4		Index for OLE
PARHIT	Integer*4		Index for PAR
TRCHIT	Integer*4		Index for METH
NONHIT	Integer*4		Index for NONR

TABLE E-30. INCLUDE FILE LUNITS.EXT

Common block	Referenced by		Description
LUNITS	BLKMOD, DUMPHD, ICPRCS, NEWICS, OPBCON, OPCONC, OPICON, OPSTAV, OPWRCN, POBCON, POBTRK, POCONC, POICON, POMXBM, POSTAV, RDBCON, RDBT, RDCONC, RDHDBM, RDICON, RDMXBM, RDSTAV, RUNMGR, WRCONC, WRSTAV		UNIT numbers of input and output files
Variables	**Type**	**Value**	**Description**
NUMBMF	Integer*4	6	Maximum number of multiple BMAT subfiles
			LUNITS
Variables	**Type**	**Dimension**	**Description**
UNITBC	Integer*4		Logical unit number of BCON file
UNITBM	Integer*4		Logical unit number of BMAT subfile
UNITBT	Integer*4		Logical unit number of BTRK file
UNITCN	Integer*4		Logical unit number of CONC file
UNITIC	Integer*4		Logical unit number of ICON file
UNITSV	Integer*4		Logical unit number of STATE VECTOR file
UNITNI	Integer*4		Logical unit number of NEW ICON file
UNITST	Integer*4		Logical unit number of STOP file
UNITPR	Integer*4		Logical unit number of PROGRESS file

TABLE E-31. INCLUDE FILE LVNAME.EXT

	Common block	Referenced by	Description
	LVNAME	BLKMOD, INIRUN	Level names for model processors
Variables	Type	Dimension	Description
LVNAME(l)	Character*4	(NLEVS)	Name of *l*th level in model order

TABLE E-32. INCLUDE FILE NDXPC.EXT

	Common block	Referenced by	Description
	NDXSPC	BCPRCS, BMPRCS, DUMPHD, ICPRCS, NEWICS, OPSTAV, ORSPBM, ORSPBC, ORSPIC, WRSTAV	Index lists for SPECIES ordering of input files
Variables	Type	Dimension	Description
NXSPIC(k)	Integer*4	(NSPECS)	Position of *k*th specie in model on ICON file
NXSPBC(k)	Integer*4	(NSPECS)	Position of *k*th specie in model on BCON file
NXSPBM(k)	Integer*4	(NSPECS)	Position of *k*th specie in model on BMAT file

TABLE E-33. INCLUDE FILE NROOTS.EXT

	Common block	Referenced by	Description
	NROOTS	HSTEP, LILGAM	HSTEP roots counts
Variables	Type	Dimension	Description
NCMPLX	Integer*4		Number of complex roots of characteristic polynomial computed
NPOS	Integer*4		Number of positive roots of characteristic polynomial computed
NDBL	Integer*4		Number of double roots of characteristic polynomial computed
NTRPL	Integer*4		Number of triple roots of characteristic polynomial computed

TABLE E-34. INCLUDE FILE RKLEVS.EXT

	Common block	Referenced by	Description
	RKLEVS	BLKMOD, RTPHO	
Variables	Type	Dimension	Description
RKL(I)	Real*4	(NRKL)	Level *l* for which photolytic rate constant is given
NRKL	Integer*4	11	Number of levels in table (from DIMENS.EXT)

TABLE E-35. INCLUDE FILE ROWSCT.EXT

	Common block	Referenced by	Description
	ROWSCT	BMPRCS, RDBMAT, RDBTRK, RDCONC, RDSTAV, RUNMGR, WRSTAV	Local row counters for I/O processes
Variables	Type	Dimension	Description
BMPSRW	Integer*4		Row counter for BMPRCS
RDBMRW	Integer*4		Row counter for RDBMAT
RDBTRW	Integer*4		Row counter for RDBTRK
RDCNRW	Integer*4		Row counter for RDCONC

TABLE E-36. INCLUDE FILE RTCONS.EXT

	Common block	Referenced by	Description
	RTCONS	BLKMOD, LILGAM, PQ1, PQCOEF, RATED, RTPHO, RTSET	Chemistry reaction and rate constants information
Variables	Type	Dimension	Description
CGA(k,l)	Real*4	(NSPECS, NLEVS)	Advection component of concentration for species *k* in level *l*
GPR(k,l)	Real*4	(NSPECS, NLEVS)	Chemistry component of concentration for species *k* in level *l*
ADTVF(k)	Real*4	(NSPECS)	Product of advection component and vertical flux component of concentration for species *k*
INVADT(k)	Real*4	(NSPECS)	1 / ADTVF(NSPECS)
RK1(m,l)	Real*4	(NRCT, NLEVS)	Rate constant for reaction *m* in level *l*
RKK1(m)	Real*4	(NRCT)	Rate constant for reaction *m*
RKTADV(m)	Real*4	(NRCT)	Product of rate constant for reaction *m* and advection component and vertical flux of concentration
PQM(m)	Real*4	(NRCT)	Product of RKTADV and chemistry component of concentration for reaction *m*
CSPR1(m,n,l)	Real*4	(5, 0:90, NRKL)	Clear sky photolytic reaction rate constant for reaction number $m = 1 - 5$, solar angles $n = 0 - 90$, rate constant level $l = 1$ - NRKL; for Carbon Bond 4.2: m = 1 corresponds to reaction number 1 m = 2 corresponds to reaction number 8 m = 3 corresponds to reaction number 33 m = 4 corresponds to reaction number 34 m = 5 corresponds to reaction number 40
NRCT	84		Number of reactions in chemistry solver (from DIMENS.EXT)
NRKL	11		Number of levels at which photolytic rate constant is given (from DIMENS.EXT)

TABLE E-37. INCLUDE FILE RTSHBC.EXT

	Common block	Referenced by	Description
	RTSHBC	RDBCON	BCON file real time step header block
Variables	Type	Dimension	Description
DATBC	Real*4		5-digit JULIAN date (*YYDDD*) on BCON file
TIMBC	Real*4		6-digit time (*HHMMSS*) on BCON file
ELPBC	Real*4		Elapsed time (seconds) since start time for BCON scenario
STPBC	Real*4		Step number on BCON file

TABLE E-38. INCLUDE FILE RTSHBM.EXT

	Common block	Referenced by	Description
	RTSHBM	RDBMAT	BMAT file real time step header block
Variables	Type	Dimension	Description
DATBM	Real*4		5-digit JULIAN date (*YYDDD*) on BMAT file
TIMBM	Real*4		6-digit time (*HHMMSS*) on BMAT file
ELPBM	Real*4		Elapsed time (seconds) since start time for BMAT scenario
STPBM	Real*4		Step number on BMAT file

TABLE E-39. INCLUDE FILE RTSHBT.EXT

	Common block	Referenced by	Description
	RTSHBT	RDBTRK	BTRK file real time step header block
Variables	Type	Dimension	Description
DATBT	Real*4		5-digit JULIAN date (*YYDDD*) on BTRK file
TIMBT	Real*4		6-digit time (*HHMMSS*) on BTRK file
ELPBT	Real*4		Elapsed time (seconds) since start time for BTRK scenario
STPBT	Real*4		Step number on BTRK file

TABLE E-40. INCLUDE FILE RTSHCN.EXT

	Common block	Referenced by	Description
	RTSHCN	RDCONC, WRCONC	CONC file real time step header block
Variables	Type	Value	Description
IDLT	Integer*4	NLEVS x NCOLS - 4	Number of words for padding time step header for file fixed record length

			RTSHCN
Variables	Type	Dimension	Description
DATCN	Real*4	(IDLT)	5-digit JULIAN date (*YYDDD*) on CONC file
TIMCN	Real*4		6-digit time (*HHMMSS*) on CONC file
ELPCN	Real*4		Elapsed time (seconds) since start time for CONC scenario
STPCN	Real*4		Step number on CONC file
RDUMCN(m)	Real*4		Dummy array for padding time step header out to record length for fixed record length file

TABLE E-41. INCLUDE FILE RTSHIC.EXT

	Common block	Referenced by	Description
	RTSHIC	NEWICS, RDICON	CONC file real time step header block
Variables	Type	Value	Description
IDLT	Integer*4	NLEVS x NCOLS - 4	Number of words for padding time step header for file fixed record length

			RTSHIC
Variables	Type	Dimension	Description
DATIC	Real*4	(IDLT)	5-digit JULIAN date (*YYDDD*) on ICON file
TIMIC	Real*4		6-digit time (*HHMMSS*) on ICON file
ELPIC	Real*4		Elapsed time (seconds) since start time for ICON scenario
STPIC	Real*4		Step number on ICON file
RDUMIC(m)	Real*4		Dummy array for padding time step header out to record length for fixed record length file

TABLE E-42. INCLUDE FILE RUNTMS.EXT

	Common block	Referenced by	Description
	RUNTMS	BIGGAM, RUNMGR, TIMER	Run timing statistics
Variables	Type	Dimension	Description
RUNCPU	Real*4		Elapsed CPU time for current run
RUNCLK	Real*4		Elapsed clock time for current run
OLDCPU	Real*4		Elapsed CPU time of prior step
OLDCLK	Real*4		Elapsed clock time of prior step
KDATE	Integer*4		Current clock date (during execution)
KTIME	Integer*4		Current clock time (during execution)

TABLE E-43. INCLUDE FILE SPNAME.EXT

	Common block	Referenced by	Description
	SPNAME	BLKMOD, DUMPHD, INIRUN, LILGAM, ORSPBC, ORSPBM, ORSPIC, PQ1	Species names list in model order
Variables	Type	Dimension	Description
SPNAME(k)	Character*4	(NSPECS)	Name of *k*th species in model order

TABLE E-44. INCLUDE FILE STOPFL.EXT

	Common block	Referenced by	Description
	STOPFG	BLKMOD, RUNMGR	Stop flag
Variables	Type	Dimension	Description
STOPFG	Character*4		Character string: .EQ. 'STOP' ⇒ model run halts at end of current time step .NE. 'STOP' ⇒ model run continues

TABLE E-45. INCLUDE FILE SUBID.EXT

Common block	Referenced by	Description
SUBID	ADATE, ASORT, BCPRCS, BIGGAM, BLKMOD, BMPRCS, BTPRCS, CELLM, CLOCK1, CLOCK2, CNPRCS, CPUTIM, DATTIM, DUMPHD, FSKIP1, GTILDE, HSTEP, ICPRCS, INDEX1, INIRUN, IOCL, JFILE2, JFILE5, JFILE6, JULIAN, JUNIT, LILGAM, NEWICS, OPBCON, OPBMAT, OPBTRK, OPCONC, OPICON, OPSTAV, OPWRCN, ORSPBC, ORSPBM, ORSPIC, POBCON, POBTRK, POCONC, POICON, POMXBM, POSTAV, PQ1, PQCOEF, PRGSMY, RATED, RDBCON, RDBMAT, RDBT, RDBTRK, RDCHAR, RDCONC, RDFILE, RDHDBM, RDICON, RDMXBM, RDSTAV, RTPHO, RTSET, RUNMGR, TIMER, WRCHAR, WRCONC, WRFILE, WRSTAV	Internal module description listed at run time by PRGSMY

Variables	Type	Dimension	Description
SUBDES	Character*80		Description of module
SUBDTE	Character*12		Date of last update of module
SUBNAM	Character*8		Module name
SUBVER	Character*8		Module version

TABLE E-46. INCLUDE FILE TEXTPT.EXT

Common block	Referenced by	Description
TEXTPT	BCPRCS, BIGGAM, BLKMOD, BMPRCS, BTPRCS, CNPRCS, ICPRCS, LILGAM, RDBCON, RDBMAT, RDBTRK, RDCONC, RDICON, RDSTAV, RUNMGR, WRCONC, WRSTAV	Text pointers for all processes

Variables	Type	Dimension	Description
BIGMPT	Integer*4		Text pointer for BIGGAM process
LILGPT	Integer*4		Text pointer for LILGAM process
BCPSPT	Integer*4		Text pointer for BCON process (BCPRCS)
BMPSPT	Integer*4		Text pointer for BMAT process (BMPRCS)
BTPSPT	Integer*4		Text pointer for BTRK process (BTPRCS)
CNPSPT	Integer*4		Text pointer for CONC process (CNPRCS)
ICPSPT	Integer*4		Text pointer for ICON process (ICPRCS)
RDBCPT	Integer*4		Text pointer for RDBCON process
RDBMPT	Integer*4		Text pointer for RDBMAT process
RDBTPT	Integer*4		Text pointer for RDBTRK process
WRCNPT	Integer*4		Text pointer for WRCONC process
RDICPT	Integer*4		Text pointer for RDICON process
RDCNPT	Integer*4		Text pointer for RDCONC process
WRSVPT	Integer*4		Text pointer for WRITE STATE VECTORS process (WRSTAV)

TABLE E-47. INCLUDE FILE TILDE.EXT

	Common block	Referenced by	Description
	TILDE	GTILDE, LILGAM, PQ1	GTILDE values
Variables	**Type**	**Dimension**	**Description**
GTILD1(k)	Real*4	(NSPECS)	GTILDE values for LEVEL ONE
GTILD2(k)	Real*4	(NSPECS)	GTILDE values for LEVEL TWO
GTILD3(k)	Real*4	(NSPECS)	GTILDE values for LEVEL THREE
GTI(k,l)	Real*4	(NSPECS, NLEVS)	GTILDE values for ONE ROW

NOTE: DIMENSION GTI(NSPECS, NLEVS)
EQUIVALENCE (GTI, GTILD1)

TABLE E-48. INCLUDE FILE TSHDBC.EXT

	Common block	Referenced by	Description
	TSHDBC	POBCON, RDBCON	BCON file time step header block
Variables	Type	Dimension	Description
IDATBC	Integer*4		5-digit JULIAN date (*YYDDD*) on BCON file
ITIMBC	Integer*4		6-digit time (*HHMMSS*) on BCON file
IELPBC	Integer*4		Elapsed time (seconds) since start time for BCON scenario
ISTPBC	Integer*4		Step number on BCON file

TABLE E-49. INCLUDE FILE TSHDBM.EXT

	Common block	Referenced by	Description
	TSHDBM	POMXBM, RDBMAT	BMAT file time step header block
Variables	Type	Dimension	Description
IDATBM	Integer*4		5-digit JULIAN date (*YYDDD*) on BMAT file
ITIMBM	Integer*4		6-digit time (*HHMMSS*) on BMAT file
IELPBM	Integer*4		Elapsed time (seconds) since start time for BMAT scenario
ISTPBM	Integer*4		Step number on BMAT file

TABLE E-50. INCLUDE FILE TSHDBT.EXT

	Common block	Referenced by	Description
	TSHDBT	POBTRK, RDBTRK	BTRK file time step header block
Variables	Type	Dimension	Description
IDATBT	Integer*4		5-digit JULIAN date (*YYDDD*) on BTRK file
ITIMBT	Integer*4		6-digit time (*HHMMSS*) on BTRK file
IELPBT	Integer*4		Elapsed time (seconds) since start time for BTRK scenario
ISTPBT	Integer*4		Step number on BTRK file

TABLE E-51. INCLUDE FILE TSHDCN.EXT

	Common block	Referenced by	Description
	TSHDCN	CNPRCS, NEWICS, POCONC, RDCONC, WRCONC	CONC file time step header block
Variables	Type	Dimension	Description
IDATCN	Integer*4		5-digit JULIAN date (*YYDDD*) on CONC file
ITIMCN	Integer*4		6-digit time (*HHMMSS*) on CONC file
IELPCN	Integer*4		Elapsed time (seconds) since start time for model scenario
ISTPCN	Integer*4		Step number on CONC file

TABLE E-52. INCLUDE FILE TSHDIC.EXT

	Common block	Referenced by	Description
	TSHDIC	CNPRCS, ICPRCS, NEWICS, POICON, RDICON	ICON file time step header block
Variables	Type	Dimension	Description
IDATIC	Integer*4		5-digit JULIAN date (*YYDDD*) on ICON file
ITIMIC	Integer*4		6-digit time (*HHMMSS*) on ICON file
IELPIC	Integer*4		Elapsed time (seconds) since start time for model scenario
ISTPIC	Integer*4		Step number on ICON file

TABLE E-53. INCLUDE FILE TSHDMD.EXT

	Common block	Referenced by	Description
	TSHDIN	BIGGAM, RUNMGR	Model time step header block
Variables	Type	Dimension	Description
IDATMD	Integer*4		5-digit JULIAN date (*YYDDD*) of current model step
ITIMMD	Integer*4		6-digit time (*HHMMSS*) of current model step
IELPMD	Integer*4		Elapsed time (seconds) since start time for model scenario
ISTPMD	Integer*4		Step number of current model step

TABLE E-54. INCLUDE FILE TSHDSV.EXT

	Common block	Referenced by	Description
	TSHDSV	POSTAV, RDSTAV, RUNMGR, WRSTAV	STATE VECTOR file time step header block
Variables	**Type**	**Dimension**	**Description**
IDATSV	Integer*4		5-digit JULIAN date (*YYDDD*) on STATE VECTOR file
ITIMSV	Integer*4		6-digit time (*HHMMSS*) on STATE VECTOR file
IELPSV	Integer*4		Elapsed time (seconds) since start time for model scenario
ISTPSV	Integer*4		Step number on STATE VECTOR file

TABLE E-55. INCLUDE FILE TSTEPS.EXT

	Common block	Referenced by	Description
	TSTEPS	BIGGAM, BCPRCS, BMPRCS, BTPRCS, ICPRCS, CNPRCS, INIRUN, RDBCON, RDBMAT, RDBTRK, RDCONC, RDICON, RDSTAV, RUNMGR, WRSTAV	Time step information for processes
Variables	**Type**	**Dimension**	**Description**
MDDATE	Integer*4		Date of current step in RUNMGR
MDTIME	Integer*4		Time of current step in RUNMGR
MDELAP	Integer*4		Elapsed time from start of scenario (seconds) in RUNMGR
MDSTEP	Integer*4		Current step number in RUNMGR
BCDATE	Integer*4		Date of current step in BCON process
BCTIME	Integer*4		Time of current step in BCON process
BCELAP	Integer*4		Elapsed time from start of scenario (seconds) in BCON process
BCSTEP	Integer*4		Current step number in BCON process
BMDATE	Integer*4		Date of current step in BMAT process
BMTIME	Integer*4		Time of current step in BMAT process
BMELAP	Integer*4		Elapsed time from start of scenario (seconds) in BMAT process
BMSTEP	Integer*4		Current step number in BMAT process
BTDATE	Integer*4		Date of current step in BTRK process
BTTIME	Integer*4		Time of current step in BTRK process
BTELAP	Integer*4		Elapsed time from start of scenario (seconds) in BTRK process
BTSTEP	Integer*4		Current step number in BTRK process
CNDATE	Integer*4		Date of current step in CONC process
CNTIME	Integer*4		Time of current step in CONC process
CNELAP	Integer*4		Elapsed time from start of scenario (seconds) in CONC process
CNSTEP	Integer*4		Current step number in CONC process
ICDATE	Integer*4		Date of current step in ICON process
ICTIME	Integer*4		Time of current step in ICON process
ICELAP	Integer*4		Elapsed time from start of scenario (seconds) in ICON process
ICSTEP	Integer*4		Current step number in ICON process

TABLE E-56. INCLUDE FILE UNITIO.EXT

Common block	Referenced by	Description
UNITIO	ADATE, BCPRCS, BIGGAM, BLKMOD, BMPRCS, BTPRCS, CELLM, CPUTIM, DATTIM, DUMPHD, FSKIP1, ICPRCS, INIRUN, JFILE2, JFILE5, JFILE6, JUNIT, LILGAM, NEWICS, OPBCON, OPBMAT, OPBTRK, OPCONC, OPICON, OPSTAV, OPWRCN, ORSPBC, ORSPBM, ORSPIC, POBCON, POBTRK, POCONC, POICON, POMXBM, POSTAV, PQ1, PRGSMY, RDBCON, RDBMAT, RDBT, RDBTRK, RDCONC, RDHDBM, RDICON, RDMXBM, RDSTAV, RUNMGR, WRCONC, WRSTAV	Logical unit numbers of system-defined default FORTRAN standard input and output files

Variables	Type	Dimension	Description
LUNIN	Integer*4		Logical unit number of standard input file
LUNOUT	Integer*4		Logical unit number of standard output file

TABLE E-57. INCLUDE FILE ZADVSL.EXT

Common block	Referenced by	Description
ZERADV	BIGGAM, BLKMOD	Positive "zero" values to replace negative advection solutions calculated by BIGGAM

Variables	Type	Dimension	Description
ZADVSL(k)	Real*4	(NSPECS)	Replacement value for species *k*

APPENDIX F

CORE MODEL ERROR CHECKING

SUBROUTINE ADATE

ERROR CHECK #1:

In subroutine ADATE, the current date (month, day, year) is obtained by calling an internal system routine. The month, day, and year are then written to an internal formatted buffer (CDATE). The I/O status value of the formatted write operation is stored in the variable IOST. If IOST is not equal to zero, then a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the month, the day, the year, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C get date from system
      CALL IDATE (IMON, IDAY, IYEAR)
      WRITE (CDATE, 1001, IOSTAT=IOST) IMON, IDAY, IYEAR
1001  FORMAT(3I2.2)
      IF (IOST .NE. 0) THEN
         WRITE (LUNOUT, 2001) IMON, IDAY, IYEAR, IOST
2001     FORMAT(/ 5X, '%%% ERROR ENCODING DATE IN ADATE'
     &          / 5X, 'IMON = ', I4, 2X, 'IDAY = ', I4,
     &          2X, 'IYEAR = ', I4, 2X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

SUBROUTINE ASORT

NONE

SUBROUTINE BCPRCS

ERROR CHECK #1:

The BCON time step header is read by calling subroutine RDBCON. The parameter IOST is passed to RDBCON. Upon return from subroutine RDBCON, IOST contains the I/O status of the read of the BCON time step header. IOST is tested; if IOST is less than zero, an end-of-file marker was reached, and control is passed back to the calling subroutine BIGGAM. If IOST is not equal to zero, then an error occurred on the read operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the time and date that were read from the BCON time step header, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read BC T.S.H.
      CALL RDBCON (IOST)
C
      IF (IOST .LT. 0) RETURN
      IF (IOST .NE. 0) THEN
         WRITE (LUNOUT, 2001) BCDATE, BCTIME, IOST
2001     FORMAT(/ 5X, '%%% READ ERROR ON T.S.H. FROM BCON FILE'
     &         / 5X, 'BCDATE = ', I6.6, 2X, 'BCTIME = ', I6.6
     &         / 5X, 'I/O STATUS = ', I8 /)
         CALL EXIT
      END IF
```

SUBROUTINE BIGGAM

ERROR CHECK #1:

Calls are made to the following subroutines:

ICPRCS to read the ICON time step header
BCPRCS to read the BCON time step header
BTPRCS to read the backtrack time step header
ICPRCS to read the initial condition concentrations data
LILGAM to write the advection time step header
BCPRCS to read the boundary conditions data
BTPRCS to read the backtrack locations and diffusivities data

A distinctive parameter is passed into each of these subroutines - the I/O status value of a read or write operation performed in the called subroutine. This parameter is tested. If it is not equal to zero, then either an end-of-file marker was reached or an error was encountered in the subroutine that was called. If the parameter is not equal to zero, then the variable IEOFBG in BIGGAM is set to a specific number but not zero. IEOFBG is set to a number determined by the routine in which the end of file was reached. IEOFBG is tested in subroutine BIGGAM. If it is not equal to zero, an error message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the value of IEOFBG. Control is returned back to the calling program RUNMGR.

CODE FOR ERROR CHECK #1:

```
C
C read BGIC T.S.H.
      CALL ICPRCS (IEOFIC)
      IF (IEOFIC .NE. 0) IEOFBG = 1
C
C read BGBC T.S.H.
      CALL BCPRCS (IEOFBC)
      IF (IEOFBC .NE. 0) IEOFBG = 2
C
C read BGBT T.S.H.
      CALL BTPRCS (IEOFBT)
      IF (IEOFBT .NE. 0) IEOFBG = 3
C
C read initial condition concentrations
      CALL ICPRCS (IEOFIC)
      IF (IEOFIC .NE. 0) IEOFBG = 4
C
C write ADVS T.S.H. (done here so that ICON's can be copied to CONC)
      CALL LILGAM (IEOFLG)
      IF (IEOFLG .NE. 0) IEOFBG = 7
C
C read boundary conditions
      CALL BCPRCS (IEOFBC)
      IF (IEOFBC .NE. 0) IEOFBG = 5
C
C read backtrack locations and diffusivities
      CALL BTPRCS (IEOFBT)
      IF (IEOFBT .NE. 0) IEOFBG = 6
C
      IF (IEOFBG .NE. 0) THEN
         WRITE (LUNOUT, 2001) IEOFBG
2001     FORMAT(// 5X, '%%% BIGGAM NOT GETTING DATA'
     &             / 2X, 'IEOFBG: ', I4 /)
         RETURN
      END IF
```

ERROR CHECK #2:

Calls are made to the following subroutines:

BCPRCS to read the BCON time step header
BTPRCS to read the backtrack time step header
ICPRCS to read the ICON file data
LILGAM to write the advection time step header
BCPRCS to read the boundary conditions data
BTPRCS to read the backtrack locations and diffusivities data

A distinctive parameter is passed into each of these subroutines - the I/O status value of a read or write operation performed in the called subroutine. This parameter is tested. If it is not equal to zero, then either an end-of-file marker was reached or an error was encountered in the subroutine that was called. If the parameter is not equal to zero, then the variable IEOFBG in BIGGAM is set to a specific number - but not to zero. IEOFBG is set to a number determined by the routine in which the end of file was reached. IEOFBG is tested in subroutine BIGGAM. If it is not equal to zero, an error message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the value of IEOFBG. Control is returned back to the calling program RUNMGR.

CODE FOR ERROR CHECK #2:

```
C
C read BGBC T.S.H.
      CALL BCPRCS (IEOFBC)
      IF (IEOFBC .NE. 0) IEOFBG = 2
C
C read BGBT T.S.H.
      CALL BTPRCS (IEOFBT)
      IF (IEOFBT .NE. 0) IEOFBG = 3
C
C read BGIC file
      CALL ICPRCS (IEOFIC)
      IF (IEOFIC .NE. 0) IEOFBG = 1
C
C write ADVS T.S.H.
      CALL LILGAM (IEOFLG)
      IF (IEOFLG .NE. 0) IEOFBG = 7
C
C read boundary conditions
      CALL BCPRCS (IEOFBC)
      IF (IEOFBC .NE. 0) IEOFBG = 5
C
C read backtrack locations and diffusivities
      CALL BTPRCS (IEOFBT)
      IF (IEOFBT .NE. 0) IEOFBG = 6
C
      IF (IEOFBG .NE. 0) THEN
         WRITE (LUNOUT, 2001) IEOFBG
         RETURN
      END IF
```

SUBROUTINE BMPRCS

ERROR CHECK #1:

The BMAT time step header is read by calling subroutine RDBMAT. The parameter IOST is passed to RDBMAT. Upon return from subroutine RDBMAT, IOST contains the read of the BMAT time step header. IOST is tested; if IOST is less than zero, then an end-of-file marker was reached and control is passed back to the calling routine BIGGAM. If IOST is not equal to zero then an error occurred on the read operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the time and date that were read from the BMAT time step header, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read BM T.S.H.
      CALL RDBMAT (IOST)
C
      IF (IOST .LT. 0) RETURN
      IF (IOST .NE. 0) THEN
         WRITE (LUNOUT, 2001) BCDATE, BCTIME, IOST
2001     FORMAT(/ 5X, '%%% READ ERROR ON T.S.H. FROM BMAT FILE'
     &          / 5X, 'BMDATE = ', I6.6, 2X, 'BMTIME = ', I6.6
     &          / 5X, 'I/O STATUS = ', I8 /)
         CALL EXIT
      END IF
```

SUBROUTINE BTPRCS

ERROR CHECK #1:

The BTRK time step header is read by calling subroutine RDBTRK. The parameter IOST is passed to RDBTRK. Upon return from subroutine RDBTRK, IOST contains the I/O status of the read operation of the BTRK time step header. IOST is tested; if IOST is less than zero, then an end-of-file marker was reached and control is passed back to the calling routine BIGGAM. If IOST is not equal to zero, then an error occurred on the read operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the time and date that were read from the BTRK time step header, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read BT T.S.H.
      CALL RDBTRK (IOST)
C
      IF (IOST .LT. 0) RETURN.
      IF (IOST .NE. 0) THEN
         WRITE (LUNOUT, 2001) BTDATE, BTTIME, IOST
2001     FORMAT(/ 5X, '%%% READ ERROR ON T.S.H. FROM BTRK FILE'
     &          / 5X, 'BTDATE = ', I6.6, 2X, 'BTTIME = ', I6.6
     &          / 5X, 'I/O STATUS = ', I8 /)
         CALL EXIT
      END IF
```

SUBROUTINE CELLM

ERROR CHECK #1:

The parameter GRDNM is passed to subroutine CELLM. GRDNM is the grid name from the standard input file (user supplied input). GRDNM is tested against the three defined grid names (NEROS1, SEROS1, ROMNET1). If GRDNM is not equal to one of these, then a message is written to the log indicating that an error has occurred. The error message contains the subroutine name and 'GRDNM'. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      DIMENSION GRDNMS(NREG)
C
      DATA GRDNMS / 'NEROS1  ', 'SEROS1  ', 'ROMNET1 ' /
C
C assume grid region is not defined
C
C assume grid region is not defined
      DO 101 IRG = 1, NREG
         IF (GRDNM .EQ. GRDNMS(IRG)) GO TO 201
101   CONTINUE
C
C grid region is not defined
      WRITE(LUNOUT, 2001) GRDNM
2001  FORMAT(// 5X, '%%% ERROR ABORT IN CELLM %%%'
     &       3X, 'GRID REGION ', A8, ' IS NOT DEFINED')
      CALL EXIT
```

SUBROUTINE CLOCK1

NONE

SUBROUTINE CLOCK2

NONE

SUBROUTINE CNPRCS

NONE

SUBROUTINE CPUTIM

NONE

SUBROUTINE DATTIM

A call to subroutine ADATE is made. The parameter DATE is passed to the subroutine ADATE. ADATE returns the current date (*MMDDYY*) from the system in the character string DATE. The parameter TIME is passed into the subroutine ADATE. ADATE returns the current time (*HHMMSS*) from the system in the character string TIME.

ERROR CHECK #1:

From the character string DATE the current month, day, and year are extracted by a formatted read operation. The I/O status value of the read operation is stored in the variable IOST. IOST is tested, and if not equal to zero, then a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the month, day, year, and I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C date
C
      CALL ADATE (DATE, TIME, TSTR)
      READ (DATE, 1001) IMON, IDAY, IYEAR
1001  FORMAT(3I2)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IMON, IDAY, IYEAR, IOST
2001     FORMAT(/ 5X, '%%% ERROR ENCODING DATE IN DATTIM '/
     &          / 5X, 'IMON =', I8, 2X, 'IDAY =', I8,
     &          2X, 'IYEAR =', I8, 3X, 'I/O STATUS = ', I2 /)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

From the character string TIME, the current hour, minute, and second are extracted by a formatted read operation. The I/O status value of the read operation is stored in the variable IOST. IOST is tested, and if not equal to zero, then a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the hour, the minute, the second, and I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C time
C
      READ (TIME, 1001, IOSTAT = IOST) IHR, IMIN, ISEC
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) IHR, IMIN, ISEC, IOST
2003     FORMAT(/ 5X, '%%% ERROR ENCODING TIME IN DATTIM '/
     &           / 5X, 'IHR =', I8, 2X, 'IMIN =', I8,
     &           2X, 'ISEC =', I8, 3X, 'I/O STATUS = ', I2 /)
         CALL EXIT
      END IF
```

SUBROUTINE DUMPHD

ERROR CHECK #1:

The BMAT file requires a large amount of disk space and may have been written to several smaller subfiles on different disk packs (since each pack may not individually have had sufficient space to contain the entire file). These subfiles would then be assigned separate logical names in the job's run stream. In subroutine RDMXBM, the subfile number is tested. If the test fails, then the B-matrix subfiles are out of order. A call to subroutine DUMPHD is made. The subfile number (ISUB) is the parameter passed to DUMPHD. Upon return from DUMPHD, the program exits by issuing a call to the system subroutine EXIT. In subroutine DUMPHD, a message is written to the log indicating that an error has occurred. The error message states that an error occurred in the BMAT sequence, and also states the subfile number, the unit number, the logical name of the file, and the actual name of the file.

CODE FOR ERROR CHECK #1:

```
C
From subroutine RDMXBM
C verify subfile
      IF (ISUBFL .NE. ISUB .OR. NSUBFL .NE. NSUB) THEN
         CALL DUMPHD (ISUB)
         CALL EXIT
      END IF
```

In subroutine DUMPHD:

```
C
C
      INQUIRE (FILE = FLNMBM(ISUB), NAME = EQNAME)
      WRITE(LUNOUT, 2001) ISUB, UNITBM, FLNMBM(ISUB), EQNAME
2001  FORMAT(/ 5X, '%%% SUBFILE INCORRECT IN BMAT SEQUENCE:'
     &       / ' SUBFILE NUMBER', I3, 3X, 'JOB ABORTED'
     &       / 5X, 'UNITBM = ', I2, 'FNAME = ', A8
     &       / 3X, 'EQNAME = ', A64)
```

ERROR CHECK #2:

An iteration over the number of species is made. For each iteration, the following takes place:

- A call to the function INDEX1 occurs. SPNAME(ISPC), and SPNMBM are passed into INDEX1. (SPNMBM is an array containing the species name in the BMAT list, obtained from the BMAT header. SPNAME is an array containing the list of species names for the model.)
- The position of each of the species names of the model is searched for in the list of species names from the BMAT list of species names. This positional value is assigned to the variable SPCNUM in the subroutine DUMPHD.
- If SPCNUM equals zero, then the species name was not found in the BMAT list of species names. A message is written to the log indicating that an error has occurred.

The error message contains the subroutine name, the current species name that is being considered (SPNAME(ISPC)), the models species name (SPNMIN(ISPC)), and the value of SPCNUM (which should be zero). The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
      DO 101 ISPC = 1, NSPCIN
C
C check that model species name is in BMATRIX species list
C and get model list species' name position in BMATRIX list
      SPCNUM = INDEX1 (SPNAME(ISPC), NSPCBM, SPNMBM)
      IF (SPCNUM .EQ. 0) THEN
         WRITE(LUNOUT, 2009) SPNAME(ISPC)
2009     FORMAT(/ 5X, 'XXX ERROR IN DUMPHD'
     &          / 5X, 'species', 2X, A4, 2X,
     &          'is not present on BMATRIX file--job aborted')
         WRITE(LUNOUT, 2011) SPNMIN(ISPC), SPNAME(ISPC), SPCNUM
2011     FORMAT(6X, A4, 8X, A4, 7X, I4.2)
         CALL EXIT
      END IF
101   CONTINUE
```

SUBROUTINE FSKIP1

This subroutine positions a file by skipping forward or backward.

ERROR CHECK #1:

The amount of records to skip is computed and stored in the variable IFORWD. An iteration of IFORWD takes place in order to skip the appropriate amount of records. For each iteration the following takes place:

- If the file is formatted, then a formatted read takes place.
- If the file is unformatted, then an unformatted read takes place. The record to be skipped is read into a dummy variable.

- The I/O status value of the read operation is stored in the variable IOST. If IOST is not equal to zero, then a read error has occurred. A message is written to the log indicating that an error has occurred.

The error message contains the subroutine name, the unit number of the file and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      DO 111 ISKP = 1, IFORWD
         IF (FMTD) READ(IDEV, 1001, IOSTAT = IOST) DUM
1001     FORMAT(A1)
         IF (.NOT. FMTD) READ(IDEV, IOSTAT = IOST) IDUM
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2001) IDEV, IOST
2001        FORMAT(/ 5X, '%%% I/O ERROR IN FSKIP'
     & / 5X, 'IDEV = ', I3, 2X, 'I/O STATUS = ', I3)
CALL EXIT
         END IF
111   CONTINUE
```

ERROR CHECK #2:

Records are skipped by moving forward through the file until an end-of-file marker is reached. For each record skipped the following takes place:

- If the file is formatted, then a formatted read takes place.
- If the file is unformatted, then an unformatted read takes place. The record to be skipped is read into a dummy variable.
- The I/O status value of the read operation is stored in the variable IOST.
- IOST is tested. If IOST is less than zero, then an end-of-file marker is reached. The file is rewound. Records are skipped and control is returned to the calling subroutine. If IOST is not equal to zero, then a read error has occurred. A message is written to the log indicating that an error has occurred.

The error message contains the subroutine name, the unit number of the file, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
201    CONTINUE
       IF (FMTD) READ(IDEV, 1001, IOSTAT = IOST) DUM
       IF (.NOT. FMTD) READ(IDEV, IOSTAT = IOST) IDUM
       IF (IOST .LT. 0) THEN
          REWIND IDEV
          DO 211 ISKP = 1, NSKIPD + IREC
IF (FMTD) READ(IDEV, 1001, IOSTAT = IOST) DUM
IF (.NOT. FMTD) READ(IDEV, IOSTAT = IOST)
211       CONTINUE
          RETURN
       END IF
       IF (IOST .NE. 0) THEN
          WRITE(LUNOUT, 2001) IDEV, IOST
          CALL EXIT
       END IF
       NSKIPD = NSKIPD + 1
       GO TO 201
```

ERROR CHECK #3:

Records are skipped by moving forward through the file until an end-of-file marker is reached. For each record skipped the following takes place:

- If the file is formatted, then a formatted read takes place.
- If the file is unformatted, then an unformatted read takes place. The record to be skipped is read into a dummy variable.
- The I/O status value of the read operation is stored in the variable IOST.
- IOST is tested. If IOST is less than zero, then an end-of-file marker is reached. Control is returned to the calling subroutine. If IOST is not equal to zero, then a read error has occurred. A message is written to the log indicating that an error has occurred.

The error message contains the subroutine name, the unit number of the file, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
301    CONTINUE
       DO 311 ISKP = 1, N2SKIP
          IF (FMTD) READ(IDEV, 1001, IOSTAT = IOST) DUM
          IF (.NOT. FMTD) READ(IDEV, IOSTAT = IOST) IDUM
          IF (IOST .LT. 0) RETURN
          IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2001) IDEV, IOST
CALL EXIT
          END IF
          NSKIPD = NSKIPD + 1
311    CONTINUE
       RETURN
       NSKIPD = NSKIPD + 1
       GO TO 301
```

SUBROUTINE GTILDE

NONE

SUBROUTINE HSTEP

NONE

SUBROUTINE ICPRCS

ERROR CHECK #1:

The ICON time step header is read by calling subroutine RDICON. The parameter IOST is passed to RDICON. Upon return from subroutine RDICON, IOST contains the I/O status of the read operation of the ICON time step header. IOST is tested; if IOST is less than zero, then an end-of-file marker was reached and control is passed back to the calling routine BIGGAM. If IOST is not equal to zero, then an error occurred on the read operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the time and date that were read from the ICON time step header, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C initialize file process elapsed time and step
      CALL CLOCK1 (ICDATE, ICTIME, SDATIC, STHRIC, TSTPIC,
     & ICELAP, ICSTEP)
C
      IF (IDATIC .NE. ICDATE  .OR.  ITIMIC .NE .ICTIME) THEN
         WRITE(LUNOUT, 2001) ICDATE, ICTIME, IDATIC, ITIMIC
2001     FORMAT(/ 5X, 'XXX DATES/TIMES DO NOT MATCH IN ICPRCS'
     &         / 5X, 'EXPECTED DATE AND TIME = ', I6.6, 5X, I6.6
     &         / 5X, 'DATE AND TIME READ     = ', I6.6, 5X, I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

A call to subroutine CLOSIC is made to close the ICON file. The parameter IOST contains the I/O status value of the close operation of the ICON file. IOST is tested. If IOST is not equal to zero, then an error has occurred on the close operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C close the ICON file
      CALL CLOSIC (IOST)
      WRITE (LUNOUT, 1001) UNITIC
1001  FORMAT(/ 5X, 'ICON file closed on unit ', I2 /)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) IOST
2003     FORMAT(/ 5X, '%%% ICON FILE CLOSE ERROR IN ICPRCS OR',
     &          ' RDICON/CLOSIC:   I/O STATUS = ', I10)
         CALL EXIT
      END IF
```

FUNCTION INDEX1

NONE

SUBROUTINE INIRUN

ERROR CHECK #1:

From the standard input file (user supplied input), the grid name is read. This is compared with the acceptable grid name obtained from the include file REGION.EXT. If these names do not match, then a message is written to the log indicating that an error has occurred. The error message contains the expected grid name and the inputted grid name. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read grid definition record
C
      READ (LUNIN, *) GRDNIN
      WRITE(LUNOUT, 1003) GRDNIN
1003  FORMAT(/ 10X, 'grid name: ', A8)
C
      IF (GRDNIN .NE. GRDNAM) THEN
         WRITE(LUNOUT, 2001) GRDNAM, GRDNIN
2001     FORMAT(/ 5X, '%%% EXPECTED REGION NAME, ', A8,
     &1X, 'DOES NOT MATCH INPUT REGION NAME, ', A8 /)
         CALL EXIT
      END IF
```

FUNCTION IOCL

NONE

FUNCTION JFILE2

ERROR CHECK #1:

This function opens a file and attaches a unit number to it. The I/O status of the open operation is stored in the variable IOST. If IOST is not equal to zero, then an error has occurred when opening the file. IOST is then passed as a parameter to function IOCL so that the clause field in the I/O status word can be extracted. Once extracted, IOST is passed back to JFILE2. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the logical name of the file, the actual name of the file, the unit number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      IF (RDONLY) THEN
         OPEN (UNIT   = IDEV,
     &         IOSTAT = IOST,
     &         FILE   = FNAME,
     &         STATUS = 'OLD',
     &         ACCESS = 'SEQUENTIAL',
     &         FORM   = FORMAT,
     &         READONLY)
         STAT = 'OLD'
         RD = 'YES'
      ELSE
         OPEN (UNIT   = IDEV,
     &         IOSTAT = IOST,
     &         FILE   = FNAME,
     &         STATUS = 'UNKNOWN',
     &         ACCESS = 'SEQUENTIAL',
     &         FORM   = FORMAT)
         STAT = 'UNKNOWN'
         RD = 'NO'
      END IF
C
      INQUIRE (FILE = FNAME, NAME = EQNAME)
C
      IF (IOST .NE. 0) THEN
         IOST  = IOCL(IOST)
         WRITE (LUNOUT, 2001) FNAME, EQNAME, IDEV, IOST
2001     FORMAT(/ 5X,'XXX ERROR ABORT IN JFILE2'
     &          / 5X, 'UNABLE TO OPEN SEQUENTIAL FILE ', A12
     &          / 5X, 'EQNAME = ', A64
     &          / 5X, 'IDEV = ', I2, 2X, 'I/O STATUS = ', I2)
         CALL EXIT
      END IF
```

FUNCTION *JFILE5*

ERROR CHECK #1:

This function opens a file and attaches a unit number to it. The I/O status of the open operation is stored in the variable IOST. If IOST is not equal to zero, then an error has occurred when opening the file. IOST is then passed as a parameter to function IOCL so that the clause field in the I/O status word can be extracted.

Once extracted IOST is passed back to JFILE5. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the logical name of the file, the actual name of the file, the unit number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
         IF (RDONLY) THEN
            OPEN (UNIT        = IDEV,
     &            IOSTAT      = IOST,
     &            FILE        = FNAME,
     &            STATUS      = 'OLD',
     &            ACCESS      = 'SEQUENTIAL',
     &            FORM        = FORMAT,
     &            RECORDTYPE  = 'FIXED',
     &            RECL        = RECLEN,
     &            READONLY)
            STAT  = 'OLD'
            RD    = 'YES'
         ELSE
            OPEN (UNIT        = IDEV,
     &            IOSTAT      = IOST,
     &            FILE        = FNAME,
     &            STATUS      = 'UNKNOWN',
     &            ACCESS      = 'SEQUENTIAL',
     &            FORM        = FORMAT,
     &            RECORDTYPE  = 'FIXED',
     &            RECL        = RECLEN)
            STAT  = 'UNKNOWN'
            RD    = 'NO'
         END IF
C
         INQUIRE (FILE = FNAME, NAME = EQNAME)
C
         IF (IOST .NE. 0) THEN
            IOST = IOCL(IOST)
            WRITE (LUNOUT, 2001) FNAME, EQNAME, IDEV, IOST
 2001       FORMAT(/ 5X,'%%% ERROR ABORT IN JFILE5'
     &             / 5X, 'UNABLE TO OPEN SEQUENTIAL FILE ', A12
     &             / '    EQNAME = ', A64
     &             / 5X, 'IDEV = ', I2, 2X, 'I/O STATUS = ', I2)
            CALL EXIT
         END IF
```

FUNCTION JFILE6

ERROR CHECK #1:

This function opens a file and attaches a unit number to it. The I/O status of the open operation is stored in the variable IOST. If IOST is not equal to zero, then an error has occurred when opening the file. IOST is then passed as a parameter to function IOCL so that the clause field in the I/O status word can be extracted. Once extracted IOST is passed back to JFILE6. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the logical name of the file, the actual name of the file, the unit number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      IF (RDONLY) THEN
         OPEN (UNIT = IDEV,
     &         IOSTAT          = IOST,
     &         FILE = FNAME,
     &         STATUS          = 'OLD',
     &         ACCESS          = 'SEQUENTIAL',
     &         FORM = FORMAT,
     &         CARRIAGECONTROL = 'FORTRAN',
     &         RECORDTYPE      = 'VARIABLE',
     &         RECL = RECLEN,
     &         READONLY)
         STAT  = 'OLD'
         RD    = 'YES'
      ELSE
         OPEN (UNIT = IDEV,
     &         IOSTAT          = IOST,
     &         FILE = FNAME,
     &         STATUS          = 'UNKNOWN',
     &         ACCESS          = 'SEQUENTIAL',
     &         FORM = FORMAT,
     &         CARRIAGECONTROL = 'FORTRAN',
     &         RECORDTYPE      = 'VARIABLE',
     &         RECL = RECLEN)
         STAT  = 'UNKNOWN'
         RD    = 'NO'
      END IF
C
      INQUIRE (FILE = FNAME, NAME = EQNAME)
C
      IF (IOST .NE. 0) THEN
         IOST  = IOCL(IOST)
         WRITE (LUNOUT, 2001) FNAME, EQNAME, IDEV, IOST
2001     FORMAT(/ 5X,'%%% ERROR ABORT IN JFILE6'
     &          / 5X, 'UNABLE TO OPEN SEQUENTIAL FILE ', A12
     &          / '    EQNAME = ', A64
     &          / 5X, 'IDEV = ', I2, 2X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

FUNCTION JULIAN

NONE

FUNCTION JUNIT

ERROR CHECK #1:

This function returns the next available FORTRAN logical unit number. The variable IUN is a counter variable initially set to 1. Each time it is incremented the following takes place:

- IUN is tested to see if the unit number corresponding to it is available. If it is not available, then IUN is incremented and tested again.

- IUN is also compared with MAXUN. MAXUN is equal to 53 and corresponds to the maximum allowable unit number available on the system. If IUN is greater than MAXUN, then an error has occurred.

An error message is written to the log indicating that an error has occurred. The message contains the function name, the value of IUN, and a table listing the numbers 1 through 53 along with a T or a F, indicating whether that particular unit number is available or not. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      IF (IUN .GT. MAXUN) GO TO 301
C
301   CONTINUE
      WRITE(LUNOUT, 2001)
2001  FORMAT(/// 1X, 'XXX ERROR ABORT IN JUNIT XXX' /
     &       1X, 'NO MORE UNIT NUMBERS AVAILABLE FOR I/O')
      WRITE(LUNOUT, 2003) (IUN, AVAIL(IUN), IUN = 1, MAXUN)
2003  FORMAT(1X, 'AVAILABLE UNIT NUMBERS ARE: ' /
     &       3(1X, 20(I2, '-', L1, 2X) /) )
      CALL EXIT
```

SUBROUTINE LILGAM

ERROR CHECK #1:

A call to subroutine PQ1 is made. In subroutine PQ1, IOST is set equal to 1 if values for GPR or ADTVF are less than or equal to zero. Otherwise IOST is not set. (GPR is the chemistry component of concentration for a particular species in a specific level; ADTVF is the product of the advection component and the vertical flux component of concentration for a particular species.) Upon return from subroutine PQ1, IOST is tested. If IOST is not equal to zero, then an error occurred in subroutine PQ1. A message is written to the log indicating that an error occurred. The error message contains the subroutine name, the row, column, layer number, value for GTTIM, and value for CHETIM. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      CALL PQ1 (P1, Q1, LEV, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IROW, ICOL, LEV, GTTIM, CHETIM
2001     FORMAT(/ 5X, 'XXX P,Q PREDICTOR RETURN ---->'
     &          / 5X, 'IROW =', I4, 2X, 'ICOL =', I4,
     &2X, 'LEV =', I4,
     &          / 5X, 'GTTIM =', 1PE15.5, 2X, 'CHETIM =', 1PE15.5 /)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

Same as error check #1

CODE FOR ERROR CHECK #2:

```
C
      CALL PQ1 (PP1, QQ1, LEV, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) IROW, ICOL, LEV, GTTIM, CHETIM
2003     FORMAT(/ 5X, 'XXX P,Q CORRECTOR RETURN ---->'
     &          / 5X, 'IROW =', I4, 2X, 'ICOL =', I4,
     &2X, 'LEV =', I4,
     &          / 5X, 'GTTIM =', 1PE15.5, 2X, 'CHETIM =', 1PE15.5 /)
         CALL EXIT
      END IF
```

SUBROUTINE NEWICS

ERROR CHECK #1:

The NEWICON file is opened unformatted. The first segment is written to an internal buffer using a formatted write statement. The I/O status of the write operation is stored in variable IOST. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the segment number and I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C write 1st segment
      WRITE(SEG1BF, 1001, IOSTAT = IOST)
     &       CDATIC, CTIMIC, SDATIC, STHRIC, TSTPIC, FRSTIC,
     &       GRDNCN,
     &       SWLNCN, SWLTCN, NELNCN, NELTCN,
     &       DLONCN, DLATCN,
     &       NCOLCN, NROWCN, NLEVCN, NSPCCN,
     &       CDBMCN, CTBMCN,
     &       CDBTCN, CTBTCN,
     &       CDBCCN, CTBCCN,
     &       CDICCN, CTICCN,
     &       ICNTCN
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4, 8I8.8, I4.4)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ' SEGMENT 1  ', IOST
2001     FORMAT(/ 5X, 'XXX INTERNAL WRITE ERROR IN NEWICS: ', A12,
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

The unit number, buffer containing the first segment record, and IOST are passed to subroutine WRCHAR, which writes the first segment record to the NEWICON file. Upon return from subroutine WRCHAR, IOST contains the I/O status of the first segment write to the NEWICON file. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number and I/O status. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
      CALL WRCHAR (UNITN1, SEG1BF, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) UNITN1, IOST
2003     FORMAT(/ 5X, '%%% HEADER WRITE ERROR IN NEWICS',
     &          / 5X, 'UNIT NUMBER = ', I2, 2X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The species names record is written to an internal buffer using a formatted write statement. The I/O status of the write operation is stored in variable IOST. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C write the species names
      WRITE(SPNMBF, 1003, IOSTAT = IOST)
     &        (SPNMCN(ISPC), ISPC = 1, NSPCCN)
1003  FORMAT( 35(A4) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ' SPEC. NAMES', IOST
         CALL EXIT
      END IF
```

ERROR CHECK #4:

The unit number, the buffer containing the species names record, and IOST are passed to subroutine WRCHAR, which writes the species names record to the NEWICON file. Upon return from the subroutine WRCHAR, IOST contains the I/O status of the write of the species names record to the NEWICON file. IOST is tested, and if not equal to zero, then a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
      CALL WRCHAR (UNITNI, SPNMBF, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2005) IOST
2005     FORMAT(/ 5X, '%%% SPECIES NAMES WRITE ERROR IN NEWICS: ',
     &2X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #5:

The layer names record is written to an internal buffer using a formatted write statement. The I/O status of the write operation is stored in variable IOST. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C write the level names
      WRITE(LEVNBF, 1005, IOSTAT = IOST)
     &        (LVNMCN(ILEV), ILEV = 1, NLEVCN)
1005  FORMAT( 3(A4) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) '  LEV. NAMES', IOST
         CALL EXIT
      END IF
```

ERROR CHECK #6:

The unit number, the buffer containing the layer names record, and IOST are passed to subroutine WRCHAR, which writes the level names record to the NEWICON file. Upon return from subroutine WRCHAR, IOST contains the I/O status of the write of the layer names record to the NEWICON file. IOST is tested, and if not equal to zero, then a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
      CALL WRCHAR (UNITNI, LEVNBF, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2007) IOST
2007     FORMAT(/ 5X, '%%% LEVEL NAMES WRITE ERROR IN NEWICS: ',
     &2X, 'I/O STATUS = ', I4)
      END IF
```

ERROR CHECK #7:

The text records are written by subroutine WRCHAR to the NEWICON file. The I/O status of the write operation is stored in variable IOST. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #7:

```
C
C write file text group
      DO 101 ITXT = 1, ICNTCN
        CALL WRCHAR (UNITNI, TEXTCN(ITXT), IOST)
        IF (IOST .NE. 0) THEN
          WRITE(LUNOUT, 2009) IOST
2009      FORMAT(/ 5X, '%%% TEXT WRITE ERROR IN NEWICS: ',
     &2X, 'I/O STATUS = ', I4)
          CALL EXIT
        END IF
101   CONTINUE
```

ERROR CHECK #8:

The NEWICON time step header is written by calling subroutine WRFILE. The unit number, number of words, starting address of the common block RTSHIC, and the variable IOST are passed to subroutine WRFILE. Upon return from subroutine WRFILE, IOST contains the I/O status value of the write of the NEWICON time step header. IOST is tested; if IOST does not equal zero, an error occurred while writing the time step header. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the I/O status value and the number of words in the buffer to be written. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #8:

```
C
C write NI T.S.H.
      CALL WRFILE (UNITNI, IWDLTH, DATIC, IOST)
      IF (IOST .NE. 0) THEN
        WRITE(LUNOUT, 2013) UNITNI, IOST, IWDLTH
2013    FORMAT(/ 5X, '%%% T.S.H. WRITE ERROR IN NEWICS',
     &          / 5X, 'UNIT NUMBER = ', I2, 2X, 'I/O STATUS = ', I4,
     &          / 5X, 'NO. OF WORDS = ', I4)
        CALL EXIT
      END IF
```

ERROR CHECK #9:

Each row of the NEWICON file is written by iterating over the chemical species. For each iteration, a call is made to subroutine WRFILE. The unit number, number of words, starting address of the common block ICFILE, and the variable IOST are passed to WRFILE. Upon return from the subroutine call to WRFILE, IOST contains the I/O status value of a write operation to the NEWICON file. IOST is tested; if IOST is not equal to zero, then an error occurred on the write operation. A message is written to the log to indicate an error has occurred. The error message contains the subroutine name, the unit number, the I/O status value and the number of words to be written. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #9:

```
C
C write NEWICON file
      DO 205 ISPC = 1, NSPECS
        CALL WRFILE(UNITNI, IWDLTH, ICFILE(1,1,ISPC),IOST)
        IF (IOST .NE. 0) THEN
          WRITE(LUNOUT, 2015) UNITNI, IOST, IWDLTH
2015      FORMAT(/ 5X, 'XXX ROW WRITE ERROR IN NEWICS',
     &           / 5X, 'UNIT NUMBER = ', I2, 2X, 'I/O STATUS = ', I4,
     &           / 5X, 'NO. OF WORDS = ', I4)
          CALL EXIT
        END IF
205   CONTINUE
```

SUBROUTINE OPBCON

ERROR CHECK #1:

The BCON file is opened. The first segment record is read from the BCON file into a buffer by calling subroutine RDCHAR. The unit number, buffer, and IOST are passed to RDCHAR. Upon return from subroutine RDCHAR, IOST contains the I/O status value of the read of the BCON first segment record. IOST is tested; if IOST does not equal zero, an error occurred on the read operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read 1st segment
      CALL RDCHAR (UNITBC, SEG1BF, IOST)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'FIRST RECORD    '
2001     FORMAT(/ 5X, 'XXX READ ERROR IN OPBCON'
     &5X, 'IOSTAT = ', I4, 4X, A16)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

Next, a formatted read from the buffer that contains the first segment record is made, and the common block HEADBC is loaded. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if

IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C convert character header to mixed char & numeric
       READ(SEG1BF, 1001, IOSTAT = IOST)
     &     CDATBC, CTIMBC, SDATBC, STHRBC, TSTPBC, FRSTBC,
     &     GRDNBC,
     &     SWLNBC, SWLTBC, NELNBC, NELTBC,
     &     DLONBC, DLATBC,
     &     NCOLBC, NROWBC, NLEVBC, NSPCBC, ICNTBC
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 5I4.4)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'INTERNAL READ #1'
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The next record read from the BCON file is the species names record. This record is read into a buffer by calling subroutine RDCHAR. The unit number, buffer, and IOST are passed to RDCHAR. A formatted read from the buffer that contains the species names record is made, and the common block HEADBC is loaded. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C read the species names record
      CALL RDCHAR (UNITBC, SPNMBF, IOST)
      READ(SPNMBF, 1003, IOSTAT = IOST)
     &          (SPNMBC(ISPC), ISPC = 1, NSPCBC)
1003  FORMAT(6(10(A4))/)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'SPECIES NAMES   '
         CALL EXIT
      END IF
```

ERROR CHECK #4:

The next record read from the BCON file is the layer names record. This record is read into a buffer by calling subroutine RDCHAR. The unit number, buffer, and IOST are passed to RDCHAR. A formatted read from the buffer that contains the layer names record is made, and the common block HEADBC is loaded. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read level names record
      CALL RDCHAR (UNITBC, LEVNBF, IOST)
      READ(LEVNBF, 1005, IOSTAT = IOST)
     &        (LVNMBC(ILEV), ILEV = 1, NLEVBC)
1005  FORMAT(2(10(A4))/)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'LEVEL NAMES     '
         CALL EXIT
      END IF
```

ERROR CHECK #5:

The next records read are the text segment records. These records are read into a buffer by iterating over the number of text records. For each iteration, a call to subroutine RDCHAR is made. The unit number, buffer, and IOST are passed to RDCHAR. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log to indicate an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C read file text group
      DO 101 ITXT = 1, ICNTBC
         CALL RDCHAR (UNITBC, TEXTBF, IOST)
C
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2001) IOST, 'TEXT RECORDS    '
CALL EXIT
         END IF
C
         TEXTBC(ITXT) = TEXTBF
101   CONTINUE
```

SUBROUTINE OPBMAT

NONE

SUBROUTINE OPBTRK

NONE

SUBROUTINE OPCONC

ERROR CHECK #1:

The CONC file is opened. The first segment record is read from the CONC file into a buffer by calling subroutine RDCHAR. The unit number, buffer, and IOST are passed to RDCHAR. Upon return from subroutine RDCHAR, IOST contains the I/O status value of the read of the CONC first segment record. IOST is tested; if IOST does not equal zero, an error occurred on the read operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read 1st segment
      CALL RDCHAR (UNITCN, SEG1BF, IOST)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'FIRST RECORD    '
2001     FORMAT(/ 5X, '%%% READ ERROR IN OPCONC'
     &5X, 'IOSTAT = ', I4, 4X, A16)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

Next, a formatted read from the buffer that contains the first segment record is made, and the common block HEADCN is loaded. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
      READ(SEG1BF, 1001, IOSTAT = IOST)
     &     CDATCN, CTIMCN, SDATCN, STHRCN, TSTPCN, FRSTCN,
     &     GRDNCN,
     &     SWLNCN, SWLTCN, NELNCN, NELTCN,
     &     DLONCN, DLATCN,
     &     NCOLCN, NROWCN, NLEVCN, NSPCCN,
     &     CDBMCN, CTBMCN,
     &     CDBTCN, CTBTCN,
     &     CDBCCN, CTBCCN,
     &     CDICCN, CTICCN,
     &     ICNTCN
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4, 8I8.8, I4.4)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'INTERNAL READ #1'
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The next record read from the CONC file is the layer names record. This record is read into a buffer by calling subroutine RDCHAR. The unit number, buffer, and IOST are passed into RDCHAR. A formatted read from the buffer that contains the level names record is made, and the common block HEADCN is loaded. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
      CALL RDCHAR (UNITCN, LEVNBF, IOST)
      READ(LEVNBF, 1005, IOSTAT = IOST)
     &           (LVNMCN(ILEV), ILEV = 1, NLEVCN)
1005  FORMAT( <NLEVS>(A4) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'LEVEL NAMES        '
         CALL EXIT
      END IF
```

ERROR CHECK #4:

The next records read are the text segment records. These records are read into a buffer by iterating over the number of text records. For each iteration, a call to subroutine RDCHAR is made. The unit number, buffer, and IOST are passed to RDCHAR. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read file text group
      DO 101 ITXT = 1, ICNTCN
        CALL RDCHAR (UNITCN, TEXTBF, IOST)
        IF (IOST .NE. 0) THEN
            WRITE(LUNOUT, 2001) IOST, 'TEXT RECORDS    '
            CALL EXIT
        END IF
101    CONTINUE
```

SUBROUTINE OPICON

ERROR CHECK #1:

The ICON file is opened. The first segment record is read from the ICON file into a buffer by calling subroutine RDCHAR. The unit number, buffer, and IOST are passed to RDCHAR. The I/O status of the read operation is stored in the variable IOST. In subroutine OPICON, IOST is tested, and if not equal to zero, then an error occurred on the read operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read 1st segment
      CALL RDCHAR (UNITIC, SEG1BF, IOST)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'FIRST RECORD    '
2001     FORMAT(/ 5X, 'XXX READ ERROR IN OPICON'
     &5X, 'IOSTAT = ', I4, 4X, A16)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

Next, a formatted read from the buffer that contains the first segment record is made, and the common block HEADIC is loaded. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
      READ(SEG1BF, 1001, IOSTAT = IOST)
     &     CDATIC, CTIMIC, SDATIC, STHRIC, TSTPIC, FRSTIC,
     &     GRDNIC,
     &     SWLNIC, SWLTIC, NELNIC, NELTIC,
     &     DLONIC, DLATIC,
     &     NCOLIC, NROWIC, NLEVIC, NSPCIC,
     &    (IDUM(ITXT), ITXT = 1, 8),
     &     ICNTIC
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4, 8I8, I4.4)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'INTERNAL READ #1'
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The next record read from the ICON file is the species names record. This record is read into a buffer by calling subroutine RDCHAR. The unit number, buffer, and IOST are passed into RDCHAR. A formatted read from the buffer that contains the species names record is made, and the common block HEADIC is loaded. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C read the species names record
      CALL RDCHAR (UNITIC, SPNMBF, IOST)
      READ(SPNMBF, 1003, IOSTAT = IOST)
     &(SPNMIC(ISPC), ISPC = 1, NSPCIC)
1003  FORMAT( <NSPECS>(A4) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'SPECIES NAMES    '
         CALL EXIT
      END IF
```

ERROR CHECK #4:

The next record read from the ICON file is the layer names record. This record is read into a buffer by calling subroutine RDCHAR. The unit number, buffer, and IOST are passed into RDCHAR. A formatted read from the buffer that contains the level names record is made and the common block HEADIC is loaded. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read the level names record
      CALL RDCHAR (UNITIC, LEVNBF, IOST)
      READ(LEVNBF, 1005, IOSTAT = IOST)
     &(LVNMIC(ILEV), ILEV = 1, NLEVIC)
1005  FORMAT( <NLEVS>(A4) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) IOST, 'LEVEL NAMES     '
         CALL EXIT
      END IF
```

ERROR CHECK #5:

The next records read are the text segment records. These records are read into a buffer by iterating over the number of text records. For each iteration, a call to subroutine RDCHAR is made. The unit number, buffer, and IOST are passed to RDCHAR. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C read file text group
      DO 101 ITXT = 1, ICNTIC
         CALL RDCHAR (UNITIC, TEXTBF, IOST)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2001) IOST, 'TEXT RECORDS    '
CALL EXIT
         END IF
C
      TEXTIC(ITXT) = TEXTBF
101   CONTINUE
```

SUBROUTINE OPSTAV

ERROR CHECK #1:

The STATE VECTOR file is opened. IOST contains the I/O status of the open operation. IOST is tested; if IOST is not equal to zero, then an error occurred while opening the file. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C open the STATE VECTOR file
C
      UNITSV = JUNIT()
      OPEN (UNITSV,
     &      FILE = FLNMSV,
     &      ACCESS = 'SEQUENTIAL',
     &      STATUS = 'UNKNOWN',
     &      IOSTAT = IOST)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) UNITSV, IOST
2001     FORMAT(/ 5X, '%%% SV FILE OPEN ERROR IN OPSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

A formatted read of the second segment header record of the STATE VECTOR file is made and the common block HEADSV is loaded. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C read STATE VECTOR header segment 2
C
      READ(UNITSV, FMT = 1003, IOSTAT = IOST)
     &     NELTSV, DLONSV, DLATSV, NCOLSV,
     &     NROWSV, NLEVSV, NSPCSV, ICNTSV
1003  FORMAT(1X, F8.3, 2(1X, F8.5), 5(1X, I4))
      IF (ICNTSV .EQ. 0) ICNTSV = 1
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) UNITSV, RECNSV, IOST
2003     FORMAT(/ 5X, '%%% SV HEADER READ ERROR IN OPSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

A formatted read of the first segment of the species names record of the STATE VECTOR file is made, and the common block CHARSV is loaded. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C read species names records
C
      READ(UNITSV, FMT = 1005, IOSTAT = IOST)
     &    (SPNMSV(ISPC), ISPC = 1, 15)
1005  FORMAT(1X, 15(A4, 1X))
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2005) UNITSV, RECNSV, IOST
2005     FORMAT(/ 5X, 'XXX SPECIES NAMES READ ERROR IN OPSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         WRITE(LUNOUT, 2005) UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #4:

A formatted read of the second segment of the species names record of the STATE VECTOR file is made, and the common block CHARSV is loaded. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
      READ(UNITSV, FMT = 1005, IOSTAT = IOST)
     &    (SPNMSV(ISPC), ISPC = 16, NSPCSV)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2005) UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #5:

A formatted read of the layer names record of the STATE VECTOR file is made, and the common block CHARSV is loaded. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C read level names record
C
      RECNSV = RECNSV + 1
      READ(UNITSV, FMT = 1005, IOSTAT = IOST)
     &    (LVNMSV(ILEV), ILEV = 1, NLEVSV)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2007) UNITSV, RECNSV, IOST
2007     FORMAT(/ 5X, '%%% LEVEL NAMES READ ERROR IN OPSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #6:

The text segment records are read by iterating over the number of text records. For each iteration, a formatted read of a text record of the STATE VECTOR file is made, and the common block CHARSV is loaded. The I/O status of each read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
      DO 103 ITXT = 1, ICNTSV
         RECNSV = RECNSV + 1
         READ(UNITSV, FMT = 1007, IOSTAT = IOST) TEXTSV(ITXT)
1007     FORMAT(1X, A80)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2003) UNITSV, RECNSV, IOST
CALL EXIT
         END IF
103   CONTINUE
```

ERROR CHECK #7:

A formatted read from the STATE VECTOR file unit number loads the first header segment record into the common block HEADIN. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #7:

```
C
C read HEADIN header record segment 1
C
      READ(UNITSV, FMT = 1001, IOSTAT = IOST)
     &     CDATIN, CTIMIN, SDATIN, STHRIN,
     &     TSTPIN, FRSTIN, GRDNIN, SWLNIN,
     &     SWLTIN, NELNIN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2009) UNITSV, RECNSV, IOST
2009     FORMAT(/ 5X, '%%% HEADIN READ ERROR IN OPSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #8:

A formatted read from the STATE VECTOR file unit number loads the second header segment record (species index records) into the common block HEADIN. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #8:

```
C
C read HEADIN header record segment 2
C
      READ(UNITSV, FMT = 1003, IOSTAT = IOST)
     &     NELTIN, DLONIN, DLATIN, NCOLIN,
     &     NROWIN, NLEVIN, NSPCIN, ICNTIN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2011) UNITSV, RECNSV, IOST
2011     FORMAT(/ 5X, '%%% HEADIN INTERNAL READ ERROR IN OPSTAV '
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #9:

A formatted read from the STATE VECTOR file unit number loads the chemistry control records into the common block CHEMIN. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #9:

```
C
      RECNSV = RECNSV + 1
      READ(UNITSV, FMT = 1015, IOSTAT = IOST)
     &    (ISPEC(ISPC), ISPC = 1, NCOUT), ULIM, BLIM, FNOLIM
1015  FORMAT(1X, <NCOUT>(I4.3, 1X), 3(E10.3, 1X) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2013) UNITSV, RECNSV, IOST
2013     FORMAT(/ 5X, 'XXX CHEMIN READ ERROR IN OPSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #10:

A formatted read from the STATE VECTOR file unit number loads the ICON species index records into the common block NDXSPC. The I/O status of the read operation is set to be the variable IOST. IOST is tested; if IOST does not equal zero, a read error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #10:

```
C
      READ(UNITSV, FMT = 1019, IOSTAT = IOST)
     &    (NXSPIC(ISPC), ISPC = 16, NSPECS)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2015) UNITSV, RECNSV, IOST
2015     FORMAT(/ 5X, 'XXX SPECIES ORDER LIST READ ERROR IN OPSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

SUBROUTINE OPWRCN

ERROR CHECK #1:

A formatted write of the common blocks CHARCN and HEADCN is made to a buffer. CHARCN and HEADCN contain the first segment record of the CONC file. The I/O status of the write operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C write 1st segment
      WRITE(SEG1BF, 1001, IOSTAT = IOST)
     &      CDATCN, CTIMCN, SDATCN, STHRCN, TSTPCN, FRSTCN,
     &      GRDNCN,
     &      SWLNCN, SWLTCN, NELNCN, NELTCN,
     &      DLONCN, DLATCN,
     &      NCOLCN, NROWCN, NLEVCN, NSPCCN,
     &      CDBMCN, CTBMCN,
     &      CDBTCN, CTBTCN,
     &      CDBCCN, CTBCCN,
     &      CDICCN, CTICCN,
     &      ICNTCN
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4, 8I8.8, I4.4)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ' SEGMENT 1  ', IOST
2001     FORMAT(/ 5X, '%%% INTERNAL WRITE ERROR IN OPWRCN: ', A12,
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

The buffer containing the first segment record is written to the CONC file by calling subroutine WRCHAR. The unit number, buffer, and IOST are passed to WRCHAR. Upon return from subroutine WRCHAR, IOST contains the I/O status value of the write of the CONC first segment header record. IOST is tested; if IOST does not equal zero, then an error occurred on the write operation. A message is written to the log to indicate an error has occurred. The error message contains the subroutine name and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
      CALL WRCHAR (UNITCN, SEG1BF, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) UNITCN, IOST
2003     FORMAT(/ 5X, '%%% HEADER WRITE ERROR IN OPWRCN'
     &          / 5X, 'UNIT NUMBER = ', I2, 2X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

A formatted write of the common block CHARCN is made to a buffer. CHARCN contains the species names record of the CONC file. The I/O status of the write operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C write the species names
      WRITE(SPNMBF, 1003, IOSTAT = IOST)
     &       (SPNMCN(ISPC), ISPC = 1, NSPCCN)
1003  FORMAT( <NSPECS>(A4) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ' SPEC. NAMES', IOST
         CALL EXIT
      END IF
C
```

ERROR CHECK #4:

The buffer containing the species names record is written to the CONC file by calling subroutine WRCHAR. The unit number, buffer, and IOST are passed to WRCHAR. Upon return from subroutine WRCHAR, IOST contains the I/O status value of the write of the CONC species names record. IOST is tested; if IOST does not equal zero, then an error occurred on the write operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
      CALL WRCHAR (UNITCN, SPNMBF, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2005) IOST
2005     FORMAT(/ 5X, 'XXX SPECIES NAMES WRITE ERROR IN OPWRCN: ',
     &2X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #5:

A formatted write of the common block CHARCN is made to a buffer. CHARCN contains the level names record of the CONC file. The I/O status of the write operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C write the level names
      WRITE(LEVNBF, 1005, IOSTAT = IOST)
     &        (LVNMCN(ILEV), ILEV = 1, NLEVCN)
1005  FORMAT( <NLEVS>(A4) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) '  LEV. NAMES', IOST
         CALL EXIT
      END IF
```

ERROR CHECK #6:

The buffer containing the level names record is written to the CONC file by calling subroutine WRCHAR. The unit number, buffer, and IOST are passed to WRCHAR. Upon return from subroutine WRCHAR, IOST contains the I/O status value of the write of the CONC level names record. IOST is tested; if IOST does not equal zero, then an error occurred on the write operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
      CALL WRCHAR (UNITCN, LEVNBF, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2007) IOST
2007     FORMAT(/ 5X, '%%% LEVEL NAMES WRITE ERROR IN OPWRCN: ',
     &2X, 'I/O STATUS = ', I4)
      END IF
```

ERROR CHECK #7:

The first text record is written to the CONC file by calling subroutine WRCHAR. The unit number, buffer, and IOST are passed to WRCHAR. Upon return from subroutine WRCHAR, IOST contains the I/O status value of the write of the first CONC text record. IOST is tested; if IOST does not equal zero, then an error occurred on the write operation. The I/O status of the write operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, a write error has occurred. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #7:

```
C
C copy input text records to CONC file
      TEXTCN(1) = 'ROM2.1  '
      CALL WRCHAR (UNITCN, TEXTCN(1), IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2009) IOST
2009     FORMAT(/ 5X, '%%% TEXT WRITE ERROR IN OPWRCN: ',
     &2X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #8:

The remaining text records are written to the CONC file by iterating over the number of text records. For each iteration a call to subroutine WRCHAR is made. The unit number, buffer, and IOST are passed to WRCHAR. Upon return from subroutine WRCHAR, IOST contains the I/O status value of the write of the CONC text records. IOST is tested; if IOST does not equal zero, then an error occurred on the write operation. A message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #8:

```
C
      DO 103 ITXT = 2, ICNTCN
         TEXTCN(ITXT) = TEXTIN(ITXT - 1)
C
         CALL WRCHAR (UNITCN, TEXTCN(ITXT), IOST)
         IF (IOST .NE. 0) THEN
            WRITE(LUNOUT, 2009) IOST
            CALL EXIT
         END IF
103   CONTINUE
```

SUBROUTINE ORSPBC

ERROR CHECK #1:

An iteration over the number of species is made. For each iteration, the following takes place:

- A call to the function INDEX1 occurs. SPNAME(ISPC) and SPNMBC are passed into INDEX1. (SPNMBC is an array containing the species name in the BCON list, obtained from the BCON header. SPNAME is an array containing the list of species names for the model.)
- The position of each of the species names of the model is searched for in the list of species names from the BCON list of species names. This positional value is assigned to the variable SPCNUM.
- If SPCNUM equals zero, then the species name was not found in the BCON list of species names. A message is written to the log indicating that an error has occurred.

The error message contains the subroutine name, the current species name that is being considered (SPNAME(ISPC)), the models species name (SPNMIN(ISPC)), and the value of SPCNUM (which should be zero). The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      DO 101 ISPC = 1, NSPCIN
C
C check that model species name is in BCON species list
C and get model list species' name position in BCON list
C
      SPCNUM = INDEX1 (SPNAME(ISPC), NSPCBC, SPNMBC)
      IF (SPCNUM .EQ. 0) THEN
         WRITE(LUNOUT, 2001) SPNAME(ISPC)
2001     FORMAT(/ 5X, '%%% ERROR IN  ORSPBC' /
     &           / 5X, 'SPECIES', 2X, A4, 2X,
     &           'IS NOT PRESENT ON BCON FILE --JOB ABORTED')
         WRITE(LUNOUT, 1003) SPNMIN(ISPC), SPNAME(ISPC), SPCNUM
1003     FORMAT(6X, A4, 8X, A4, 7X, I4.2)
         CALL EXIT
      END IF
101   CONTINUE
```

SUBROUTINE ORSPBM

ERROR CHECK #1:

An iteration over the number of species is made. For each iteration the following takes place:

- A call to the function INDEX1 occurs. SPNAME(ISPC) and SPNMBM are passed into INDEX1. (SPNMBM is an array containing the species name in the BMAT list, obtained from the BMAT header. SPNAME is an array containing the list of species names for the model.)
- The position of each of the species names of the model is searched for in the list of species names from the BMAT list of species names. This positional value is assigned to the variable SPCNUM.
- If SPCNUM equals zero, then the species name was not found in the BMAT list of species names. A message is written to the log to indicate that an error has occurred.

The error message contains the subroutine name, the current species name that is being considered (SPNAME(ISPC)), the models species name (SPNMIN(ISPC)), and the value of SPCNUM (which should be zero). The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      DO 101 ISPC = 1, NSPCIN
C
C check that model species name is in BMATRIX species list
C and get model list species' name position in BMATRIX list
C
      SPCNUM = INDEX1 (SPNAME(ISPC), NSPCBM, SPNMBM)
        IF (SPCNUM .EQ. 0) THEN
          WRITE(LUNOUT, 2001) SPNAME(ISPC)
2001      FORMAT(/ 5X, 'XXX ERROR IN ORSPBM'
     &           / 5X, 'SPECIES', 2X, A4, 2X,
     &           'IS NOT PRESENT ON BMATRIX FILE--JOB ABORTED')
          WRITE(LUNOUT, 1003) SPNMIN(ISPC), SPNAME(ISPC), SPCNUM
1003      FORMAT(6X, A4, 8X, A4, 7X, I4.2)
          CALL EXIT
        END IF
C
101   CONTINUE
```

SUBROUTINE ORSPIC

ERROR CHECK #1:

An iteration over the number of species is made. For each iteration the following takes place:

- A call to the function INDEX1 occurs. SPNAME(ISPC) and SPNMIC are passed into INDEX1. (SPNMIC is an array containing the species name in the ICON list, obtained from the ICON header. SPNAME is an array containing the list of species names for the model.)
- The position of each of the species names of the model is searched for in the list of species names from the ICON list of species names. This positional value is assigned to the variable SPCNUM.
- If SPCNUM equals zero, then the species name was not found in the ICON list of species names. A message is written to the log to indicate that an error has occurred.

The error message contains the subroutine name, the current species name that is being considered (SPNAME(ISPC)), the models species name (SPNMIN(ISPC)), and the value of SPCNUM (which should be zero). The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
      DO 101 ISPC = 1, NSPCIN
C check that model species name is in ICON species list
C and get model list species' name position in ICON list
C
      SPCNUM = INDEX1 (SPNAME(ISPC), NSPCIC, SPNMIC)
      IF (SPCNUM .EQ. 0) THEN
         WRITE(LUNOUT, 2001) SPNAME(ISPC)
2001     FORMAT(/ 5X, '%%% ERROR IN ORSPIC' /
     &          / 5X, 'SPECIES', 2X, A4, 2X,
     &          'IS NOT PRESENT ON ICON FILE--JOB ABORTED')
         WRITE(LUNOUT, 1003) SPNMIN(ISPC), SPNAME(ISPC), SPCNUM
1003     FORMAT(6X, A4, 8X, A4, 7X, I4.2)
         CALL EXIT
      END IF
```

SUBROUTINE POBCON

ERROR CHECK #1:

A call to subroutine CLOCK1 is made to obtain the file process elapsed time step. The first time step (FRSTBC) is compared with the elapsed time step (IELPBC) obtained from the call to CLOCK1. If the times do not match, then an error has occurred. The error message contains the subroutine in which the error occurred, the expected time, the time read from the time step header, the scenario start time, the elapsed time, the step number, and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C compute intervals and records to be skipped
C
      CALL CLOCK1 (IDATE, ITIME, SDATBC, STHRBC, TSTPBC,
     & IELPBC, ISTPBC)
C
      IF (IELPBC .LT. FRSTBC) THEN
         WRITE(LUNOUT, 2001) IDATE, ITIME, SDATBC, STHRBC,
     &            IELPBC, ISTPBC, TSTPBC
2001     FORMAT(/ 5X, '%%% DATE/TIME PRECEDES FIRST DATE/TIME ON ',
     &'BCON FILE IN POBCON'
     &          / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     &          / 5X, 'SCENARIO START:      ', I5, 3X, I4
     &          / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
     &          / 5X, 'TIME STEP SIZE: ', I4 /)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

A call to FSKIP1 is made in order to skip the appropriate amount of records. The parameter IOST is passed to FSKIP1. Upon return from the subroutine FSKIP1, IOST contains the I/O status value of the read operation. IOST is tested; if IOST is equal to zero, then an end-of-file marker was reached while reading records on the

BCON file. The error message contains the subroutine in which the error occurred, the requested time, the scenario start time, the elapsed time, the step number, the I/O status value and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C and skip forward -
C
      CALL FSKIP1 (UNITBC, FMTD, SKIPDR, NSKIP, RECPOS, SKIPNO,
     & IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) IDATE, ITIME, SDATBC, STHRBC,
     &             IELPBC, ISTPBC, IOST, TSTPBC
2003     FORMAT(/ 5X, '%%% ERROR ENCOUNTERED ON BCON FILE',
     &          ' BEFORE DATE/TIME REACHED IN POBCON'
     &          / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     &          / 5X, 'SCENARIO START:      ', I5, 3X, I4
     &          / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
     &          / 5X, 'IOSTAT: ', I8, 4X, 'TIME STEP SIZE: ', I4 /)
         CALL EXIT
      END IF
```

SUBROUTINE POBTRK

ERROR CHECK #1:

A call to subroutine CLOCK1 is made to obtain the file process elapsed time step. The first time step (FRSTBT) is compared with the elapsed time step (IELPBT) obtained from the call to CLOCK1. If the times do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine in which the error occurred, the expected time, the time read from the time step header, the scenario start time, the elapsed time, the step number, and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C determine and validate step number
C
      CALL CLOCK1 (IDATE, ITIME, SDATBT, STHRBT, TSTPBT,
     & IELPBT, ISTPBT)
C
      IF (IELPBT .LT. FRSTBT) THEN
         WRITE(LUNOUT, 2001) IDATE, ITIME, SDATBT, STHRBT,
     &             IELPBT, ISTPBT, TSTPBT
2001     FORMAT(/ 5X, '%%% DATE/TIME PRECEDES FIRST DATE/TIME ON ',
     &'BTRK FILE IN POBTRK'
     &          / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     &          / 5X, 'SCENARIO START:      ', I5, 3X, I4
     &          / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
     &          / 5X, 'TIME STEP SIZE: ', I4 /)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

To skip the appropriate amount of records an iteration over the number of records is done. For each iteration a call to RDBT is made. The parameter IOST is passed to RDBT. Upon return from subroutine RDBT, IOST contains the I/O status value of a read operation. If IOST is not equal to zero, then an error occurred while reading records on the BTRK file. A message is written to the log to indicate an error has occurred. The error message contains the subroutine in which the error occurred, the expected time, the time read from the time step header, the scenario start time, the elapsed time, the step number, and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
      DO 101 ISKIP = 1, NSKIP
         CALL RDBT (1, ADUM, IOST)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2003) IDATE, ITIME, SDATBT, STHRBT,
     &            IELPBT, ISTPBT, IOST, TSTPBT
2003       FORMAT(/ 5X, 'XXX ERROR ENCOUNTERED ON BTRK FILE',
     &      ' BEFORE DATE/TIME REACHED IN POBTRK'
     & / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     & / 5X, 'SCENARIO START:      ', I5, 3X, I4
     & / 5X, 'ELAPSED TIME: ', I7,
     &   2X, 'STEP NUMBER: ', I3
     & / 5X, 'IOSTAT: ', I8,
     &   4X, 'TIME STEP SIZE: ', I4 /)
CALL EXIT
         END IF
101   CONTINUE
```

SUBROUTINE POCONC

ERROR CHECK #1:

A call to subroutine CLOCK1 is made to obtain the file process elapsed time step. The first time step (FRSTCN) is compared with the elapsed time step (IELPCN) obtained from the call to CLOCK1. If the times do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine in which the error occurred, the expected time, the time read from the time step header, the scenario start time, the elapsed time, the step number, and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C compute step number
C
      CALL CLOCK1 (IDATE, ITIME, SDATCN, STHRCN, TSTPCN,
     & IELPCN, ISTPCN)
C
      IF (IELPCN .LT. FRSTCN) THEN
         WRITE(LUNOUT, 2001) IDATE, ITIME, SDATCN, STHRCN,
     &           IELPCN, ISTPCN, TSTPCN
2001     FORMAT(/ 5X, '%%% DATE/TIME PRECEDES FIRST DATE/TIME ON ',
     &'CONC FILE IN POCONC'
     &        / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     &        / 5X, 'SCENARIO START:      ', I5, 3X, I4
     &        / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
     &        / 5X, 'TIME STEP SIZE: ', I4 /)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

A call to FSKIP1 is made in order to skip the appropriate amount of records. The parameter IOST is passed to FSKIP1. Upon return from the subroutine FSKIP1, IOST contains the I/O status value of the read operation. IOST is tested; if IOST is equal to zero, then an end-of-file marker was reached while reading records on the CONC file. The error message contains the subroutine in which the error occurred, the requested time, the scenario start time, the elapsed time, the step number, the I/O status value and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C and skip forward -
C
      CALL FSKIP1 (UNITCN, FMTD, SKIPDR, NSKIP, RECPOS, SKIPNO,
     & IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) IDATE, ITIME, SDATCN, STHRCN,
     &           IELPCN, ISTPCN, IOST, TSTPCN
2003     FORMAT(/ 5X, '%%% ERROR ENCOUNTERED ON CONC FILE',
     &        ' BEFORE DATE/TIME REACHED IN POCONC'
     &        / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     &        / 5X, 'SCENARIO START:      ', I5, 3X, I4
     &        / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
     &        / 5X, 'IOSTAT: ', I8, 4X, 'TIME STEP SIZE: ', I4 /)
         CALL EXIT
      END IF
```

SUBROUTINE POICON

ERROR CHECK #1:

A call to subroutine CLOCK1 is made to obtain the file process elapsed time step. The first time step (FRSTIC) is compared with the elapsed time step (IELPIC) obtained from the call to CLOCK1. If the times do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine in which the error occurred, the expected time, the time read from the time step header, the scenario start time, the elapsed time, the step number, and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C compute step number
C
       CALL CLOCK1 (IDATE, ITIME, SDATIC, STHRIC, TSTPIC,
      & IELPIC, ISTPIC)
C
       IF (IELPIC .LT. FRSTIC) THEN
          WRITE(LUNOUT, 2001) IDATE, ITIME, SDATIC, STHRIC,
      &            IELPIC, ISTPIC, TSTPIC
2001      FORMAT(/ 5X, '%%% DATE/TIME PRECEDES FIRST DATE/TIME ON ',
      &'ICON FILE IN POICON'
      &          / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
      &          / 5X, 'SCENARIO START:      ', I5, 3X, I4
      &          / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
      &          / 5X, 'TIME STEP SIZE: ', I4 /)
          CALL EXIT
       END IF
```

ERROR CHECK #2:

A call to FSKIP1 is made in order to skip the appropriate amount of records. The parameter IOST is passed to FSKIP1. Upon return from the subroutine FSKIP1, IOST contains the I/O status value of the read operation. IOST is tested; if IOST is equal to zero, then an end-of-file marker was reached while reading records on the ICON file. The error message contains the subroutine in which the error occurred, the requested time, the scenario start time, the elapsed time, the step number, the I/O status value and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C and skip forward -
C
      CALL FSKIP1 (UNITIC, FMTD, SKIPDR, NSKIP, RECPOS, SKIPNO,
     & IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) IDATE, ITIME, SDATIC, STHRIC,
     &            IELPIC, ISTPIC, IOST, TSTPIC
2003     FORMAT(/ 5X, 'XXX ERROR ENCOUNTERED ON ICON FILE',
     &          ' BEFORE DATE/TIME REACHED IN POICON'
     &          / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     &          / 5X, 'SCENARIO START:      ', I5, 3X, I4
     &          / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
     &          / 5X, 'IOSTAT: ', I8, 4X, 'TIME STEP SIZE: ', I4 /)
         CALL EXIT
      END IF
```

SUBROUTINE POMXBM

ERROR CHECK #1:

A call to subroutine CLOCK1 is made. The first time step (FRSTBM) is compared with the elapsed time step (IELPBM) obtained from the call to CLOCK1. If the times do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine in which the error occurred, the expected time, the time read from the time step header, the scenario start time, the elapsed time, the step number, and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C get position step number
C
      CALL CLOCK1 (IDATE, ITIME, SDATBM, STHRBM, TSTPBM,
     & IELPBM, ISTPBM)
C
      IF (IELPBM .LT. FRSTBM) THEN
         WRITE(LUNOUT, 2001) IDATE, ITIME, SDATBM, STHRBM,
     &            IELPBM, ISTPBM, TSTPBM
2001     FORMAT(/ 5X, 'XXX DATE/TIME PRECEDES FIRST DATE/TIME ON ',
     &'BMAT FILE IN POMXBM'
     &          / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     &          / 5X, 'SCENARIO START:      ', I5, 3X, I4
     &          / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
     &          / 5X, 'TIME STEP SIZE: ', I4 /)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

The appropriate amount of records are skipped by performing an IF test. For each test, a call to subroutine RDMXBM is made. The parameter IOST is passed to RDMXBM. Upon return from subroutine RDMXBM, IOST contains the I/O status of a read of the BMAT file. A test of the correct subfile is made. If this test fails, then IOST is tested; if IOST is less than zero, then the end-of-file marker is reached on the BMAT file. If this test passes, then IOST is tested, and if not equal to zero, then an error occurred while reading the BMAT file. An error message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the requested time, the scenario start time, the elapsed time, the step number, the I/O status value, and the time step size. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
      CALL RDMXBM (1, ADUM, IOST)
      IF ( ISUBFL .NE. JSUBFL) THEN
         IF (IOST .LT. 0) THEN
WRITE(LUNOUT, 2003) '   EOF', IDATE, ITIME, SDATBM,
     &   STHRBM, IELPBM, ISTPBM, IOST, TSTPBM
2003        FORMAT(/ 5X, 'XXX', A6, ' ENCOUNTERED ON BMAT FILE'
     & ' BEFORE DATE/TIME REACHED IN POMXBM'
     &          / 5X, 'REQUESTED DATE/TIME: ', I5, 5X, I6
     &          / 5X, 'SCENARIO START:      ', I5, 3X, I4
     &          / 5X, 'ELAPSED TIME: ', I7, 2X, 'STEP NUMBER: ', I3
     &          / 5X, 'IOSTAT: ', I8, 4X, 'TIME STEP SIZE: ', I4 /)
CALL EXIT
      ELSE
        .
        .
        .
      ELSE IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) ' ERROR', IDATE, ITIME, SDATBM, STHRBM,
     &          IELPBM, ISTPBM, IOST, TSTPBM
         CALL EXIT
      END IF
```

SUBROUTINE POSTAV

ERROR CHECK #1:

A call to subroutine CLOCK1 is made to obtain the elapsed time (IELPSV). This time is tested against the first time step (FRSTSV). If the elapsed time step is less than the first time step, then an error has occurred. An error message is written to the log indicating that an error has occurred. The error message contains the subroutine in which the error occurred. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C compute step
C
       CALL CLOCK1 (IDATE, ITIME, SDATSV, STHRSV, TSTPSV,
      & IELPSV, ISTPSV)
C
.
.
.
        IF (IELPSV .LT. FRSTSV) THEN
          WRITE(LUNOUT, 2201)
2201      FORMAT(/ 5X, '%%% DATE/TIME PRECEDES FIRST DATE/TIME ON ',
      &'STATE VECTOR IN POSTAV')
          CALL EXIT
       END IF
```

ERROR CHECK #2:

To skip records on the STATE VECTOR file, an iteration is made. For each iteration, a formatted read from the STATE VECTOR file is done, and the I/O status is saved in the variable IOST. IOST is tested after each read; if IOST does not equal zero, then an error has occurred. A message is written to the log indicating that an error has occurred. The message contains the subroutine name, the unit number, the current record being read, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
       DO 101 IREC = 1, RECNSV
          READ(UNIT = UNITSV, FMT = 1001, IOSTAT = IOST) DUMBUF
1001      FORMAT(A80)
          IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2003) UNITSV, IREC, DUMBUF, IOST
2003         FORMAT(/ 5X, '%%% READ ERROR IN POSTAV'
      & / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
      & / 5X, A80
      & / 5X, 'I/O STATUS = ', I4)
CALL EXIT
          END IF
101    CONTINUE
```

SUBROUTINE PQ1

ERROR CHECK #1:

For each iteration over each species, each chemistry component (GPR) is tested to see if it is less than or equal to zero. At the same time, each ADTVF component is also tested to see if it is less than or equal to zero. If either test fails, then an error has occurred. A message is written to the log. The message contains the subroutine name, as well as some chemistry reaction and rate constant values. IOST is then set to have a value of 1, and control is returned to the calling subroutine.

CODE FOR ERROR CHECK #1:

```
C
      DO 201 ISPC = 1, NSPCIN
        IF (GPR(ISPC,LEV) .LE. 0.0 .OR. ADTVF(ISPC) .LE. 0.0) THEN
          WRITE(LUNOUT, 2001) LEV
2001      FORMAT(/ 5X, 'XXX ERROR REPORTED FROM PQ1 FOR LEVEL', I2,
     & 2X, '- ZERO OR NEGATIVE CONC COMPONENT VALUES:',
     &          / 5X, 'SPEC  --- GPR(SPEC) ---
     &    GTI(SPEC) --- CGA(SPEC)')
          DO 101 JSPC = 1, NSPCIN
WRITE(LUNOUT, 2003) SPNAME(JSPC), GPR(JSPC,LEV),
     & GTI(JSPC,LEV), CGA(JSPC,LEV)
2003        FORMAT(/ 5X, A4, 2X, 3(2X, 1PE12.5))
101       CONTINUE
          IOST = 1
          RETURN
        END IF
201   CONTINUE
```

SUBROUTINE PQCOEF

NONE

SUBROUTINE PRGSMY

ERROR CHECK #1:

A call to subroutine ADATE is made to obtain the run date and run time. A formatted read from the buffer extracts the date and time. IOST contains the I/O status of this read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log indicating that an error has occurred. The error message contains the subroutine name, the run date and run time extracted from the call to ADATE, the entire character string from ADATE, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C get run date, time
C
      CALL ADATE (CDT(1), CDT(2), DUMSTR)
      READ(STRING, 1001, IOSTAT = IOST) RUNDTE, RUNTIM
1001  FORMAT(I6, 2X, I6)
      IF (IOST .NE. 0) THEN
         IOST = IOCL (IOST)
         WRITE(LUNOUT, 2001) RUNDTE, RUNTIM, STRING, IOST
2001     FORMAT(/ 5X, 'XXX ERROR ABORT IN PRGSMY -'
     &          ' UNABLE TO DECODE DATE/TIME'
     &          / 5X, 'RUNDTE = ', I8, 2X, 'RUNTIM = ', I8
     &          / 5X, 'STRING = ', A16, 2X, 'IOST = ', I6)
         CALL EXIT
      END IF
```

SUBROUTINE RATED

NONE

SUBROUTINE RDBCON

ERROR CHECK #1:

The BCON file is opened by a call to OPBCON, and the common block HEADBC is loaded. HEADBC contains the header information for the BCON file. The header information is then tested in the subroutine RDBCON. If any of the parameters fail the test, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name and the BCON header information. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C open BCON file
      CALL OPBCON
C
C check header parameters
      IF ((TSTPBC .NE. TSTPIN) .OR. (GRDNBC .NE. GRDNIN) .OR.
     &    (ABS(SWLNBC - SWLNIN) .GT. 0.001) .OR.
          .
          .
          .
         WRITE(LUNOUT, 2001) TSTPBC, TSTPIN, GRDNBC, GRDNIN,
     &            SWLNBC, SWLNIN, SWLTBC, SWLTIN,
     &            NELNBC, NELNIN, NELTBC, NELTIN,
     &            DLONBC, DLONIN, DLATBC, DLATIN,
     &            NCOLBC, NCOLIN, NROWBC, NROWIN,
     &            NLEVBC, NLEVIN, NSPCBC, NSPCIN
2001     FORMAT(/ 5X, '%%% PARAMETER CHECK FAILURE IN RDBCON'
     &          / 5X, 'TSTPBC = ', I5, 5X, 'TSTPIN = ', I5
     &          / 5X, 'GRDNBC = ', A8, 5X, 'GRDNIN = ', A8
     &          / 5X, 'NELNBC = ', F10.5, 5X, 'NELNIN = ', F10.5
     &          / 5X, 'NELTBC = ', F10.5, 5X, 'NELTIN = ', F10.5
     &          / 5X, 'SWLNBC = ', F10.5, 5X, 'SWLNIN = ', F10.5
     &          / 5X, 'SWLTBC = ', F10.5, 5X, 'SWLTIN = ', F10.5
     &          / 5X, 'DLONBC = ', F10.5, 5X, 'DLONIN = ', F10.5
     &          / 5X, 'DLATBC = ', F10.5, 5X, 'DLATIN = ', F10.5
     &          / 5X, 'NCOLBC = ', I3, 5X, 'NCOLIN = ', I3
     &          / 5X, 'NROWBC = ', I3, 5X, 'NROWIN = ', I3
     &          / 5X, 'NLEVBC = ', I3, 5X, 'NLEVIN = ', I3
     &          / 5X, 'NSPCBC = ', I3, 5X, 'NSPCIN = ', I3 )
         CALL EXIT
      END IF
```

ERROR CHECK #2:

The BCON time step header is read by calling the subroutine RDFILE. The unit number, number of words to be read, starting address of the common block RTSHBC, and IOST are passed to RDFILE. Upon return from the call to RDFILE, IOST contains the I/O status of the read operation of the BCON time step header. IOST is tested; if IOST is less than zero, then an end-of-file marker was reached. Control is returned to the calling subroutine. If IOST is not equal to zero, then an error occurred on the read operation. A message is written to the log to indicate that an error occurred. The message contains the subroutine name, the number of words to read, the value of the I/O status, the time step date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C read BCON T.S.H.
C
      CALL RDFILE (UNITBC, NWDTSH, DATBC, IOST)
C
      IF (IOST .LT. 0) RETURN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) NWDTSH, IOST, BCDATE, BCTIME
2003     FORMAT(/ 5X, '%%% T.S.H. READ ERROR IN RDBCON'
     &          / 5X, 'NO. OF WORDS = ', I2, 2X, 'I/O STATUS = ', I4,
     &2X, 'BCDATE = ', I6.6, 2X, 'BCTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The current model time step time and date are compared with the time step time and date obtained from reading the BCON time step header. If either the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the time step header, and the model date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
      IF (BCDATE .NE. IDATBC .OR. BCTIME .NE. ITIMBC) THEN
         WRITE(LUNOUT, 2005) IDATBC, ITIMBC, BCDATE, BCTIME
2005     FORMAT(/ 5X, '%%% DATES/TIMES DO NOT MATCH IN RDBCON'
     &          / 5X, 'IDATBC = ', I6.6, 2X, 'ITIMBC = ', I6.6
     &          / 5X, 'BCDATE = ', I6.6, 2X, 'BCTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #4:

The western boundary conditions are read by iterating over the number of species. For each iteration, a call to RDFILE is made. The unit number, the number of words to read, the starting address of the common block BCFILE, and IOST are passed to RDFILE. Upon return from RDFILE, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the I/O status value, and the number of words. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read Western boundary conditions . . . . . . . . . . species*
C
      DO 201 ISPC = 1, NSPECS
         CALL RDFILE (UNITBC, ROWWRD, WEST(1,1,ISPC), IOST)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2007) UNITBC, IOST, ROWWRD
2007        FORMAT(/ 5X, '%%% READ ERROR IN RDBCON'
     & / 5X, 'UNIT = ', I3, 2X, 'I/O STATUS = ', I3,
     &   2X, 'NO. OF WORDS  = ', I4)
CALL EXIT
         END IF
201   CONTINUE
```

ERROR CHECK #5:

The eastern boundary conditions are read by iterating over the number of species. For each iteration, a call to RDFILE is made. The unit number, the number of words to read, the starting address of the common block BCFILE, and IOST are passed to RDFILE. Upon return from RDFILE, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the I/O status value, and the number of words. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C read Eastern boundary conditions . . . . . . . . . . species*
C
      DO 301 ISPC = 1, NSPECS
         CALL RDFILE (UNITBC, ROWWRD, EAST(1,1,ISPC), IOST)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2007) UNITBC, IOST, ROWWRD
CALL EXIT
         END IF
301   CONTINUE
```

ERROR CHECK #6:

The northern boundary conditions are read by iterating over the number of species. For each iteration, a call to RDFILE is made. The unit number, the number of words to read, the starting address of the common block BCFILE, and IOST are passed to RDFILE. Upon return from RDFILE, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the I/O status value, and the number of words. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
C read Northern boundary conditions . . . . . . . . . . . species*
C
      DO 401 ISPC = 1, NSPECS
         CALL RDFILE (UNITBC, COLWRD, NORTH(1,1,ISPC), IOST)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2007) UNITBC, IOST, COLWRD
CALL EXIT
         END IF
401   CONTINUE
```

ERROR CHECK #7:

The southern boundary conditions are read by iterating over the number of species. For each iteration, a call to RDFILE is made. The unit number, the number of words to read, the starting address of the common block BCFILE, and IOST are passed to RDFILE. Upon return from RDFILE, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the I/O status value, and the number of words. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #7:

```
C
C read Southern boundary conditions . . . . . . . . . . . species*
C
      DO 501 ISPC = 1, NSPECS
         CALL RDFILE (UNITBC, COLWRD, SOUTH(1,1,ISPC), IOST)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2007) UNITBC, IOST, COLWRD
CALL EXIT
         END IF
501   CONTINUE
```

SUBROUTINE RDBMAT

ERROR CHECK #1:

The BMAT file is opened by a call to OPBMAT, and the common block HEADBM is loaded. HEADBM contains the header information for the BMAT file. The header information is then tested in the subroutine RDBMAT. If any of the parameters fail the test, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name and the BMAT header information. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C open BMAT file
      CALL OPBMAT
C
C check parameters in headers
      IF ((TSTPBM .NE. TSTPIN) .OR. (GRDNBM .NE. GRDNIN) .OR.
     &    (ABS(SWLNBM - SWLNIN) .GT. 0.001) .OR.
     &    (ABS(SWLTBM - SWLTIN) .GT. 0.001) .OR.
          .
          .
          .
2001      FORMAT(/ 5X, 'XXX PARAMETER CHECK FAILURE IN RDBMAT'
     &          / 5X, 'TSTPBM = ', I5, 5X, 'TSTPIN = ', I5
     &          / 5X, 'GRDNBM = ', A8, 5X, 'GRDNIN = ', A8
     &          / 5X, 'NELNBM = ', F10.5, 5X, 'NELNIN = ', F10.5
     &          / 5X, 'NELTBM = ', F10.5, 5X, 'NELTIN = ', F10.5
     &          / 5X, 'SWLNBM = ', F10.5, 5X, 'SWLNIN = ', F10.5
     &          / 5X, 'SWLTBM = ', F10.5, 5X, 'SWLTIN = ', F10.5
     &          / 5X, 'DLONBM = ', F10.5, 5X, 'DLONIN = ', F10.5
     &          / 5X, 'DLATBM = ', F10.5, 5X, 'DLATIN = ', F10.5
     &          / 5X, 'NCOLBM = ', I3, 5X, 'NCOLIN = ', I3
     &          / 5X, 'NROWBM = ', I3, 5X, 'NROWIN = ', I3
     &          / 5X, 'NLEVBM = ', I3, 5X, 'NLEVIN = ', I3
     &          / 5X, 'NSPCBM = ', I3, 5X, 'NSPCIN = ', I3 )
          CALL EXIT
      END IF
```

ERROR CHECK #2:

The BMAT time step header is read by calling the subroutine RDMXBM. The number of words to be read, starting address of the common block RTSHBM, and IOST are passed to RDMXBM. Upon return from the call to RDMXBM, IOST contains the I/O status of the read operation of the time step header. IOST is tested; If IOST is less than zero, then an end-of-file marker was reached. Control is returned to the calling subroutine. If IOST is not equal to zero, then an error occurred on the read of the BMAT time step header. A message is written to the log indicating that an error occurred. The message contains the subroutine name, the number of words to read, the value of the I/O status, the time step date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C read BMAT T.S.H.
      CALL RDMXBM (NWDTSH, DATBM, IOST)
C
      IF (IOST .LT. 0) RETURN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) NWDTSH, IOST, BMDATE, BMTIME
2003     FORMAT(/ 5X, 'XXX T.S.H. READ ERROR IN RDBMAT'
     &          / 5X, 'NO. OF WORDS = ', I2, 2X, 'I/O STATUS = ', I4,
     &2X, 'BMDATE = ', I6.6, 2X, 'BMTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The current model time step time and date are compared with the time step time and date obtained from reading the BMAT time step header. If the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the time step header, and the model date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
      IF (IDATBM .NE. BMDATE .OR. ITIMBM .NE. BMTIME) THEN
         WRITE(LUNOUT, 2005) IDATBM, ITIMBM, BMDATE, BMTIME
2005     FORMAT(/ 5X, '%%% DATES/TIMES DO NOT MATCH IN RDBMAT'
     &         / 5X, 'IDATBM = ', I6.6, 2X, 'ITIMBM = ', I6.6
     &         / 5X, 'BMDATE = ', I6.6, 2X, 'BMTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #4:

Part 1 of the BMAT file is read by calling subroutine RDMXBM. The number of words to read, the starting address of the common block BMFILE, and IOST are passed to RDMXBM. Upon return from RDMXBM, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the I/O status value, the number of words, the model step date and time, and the row. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read BMAT part 1
      CALL RDMXBM (NWDBM1, XB12, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2007) IOST, NWDBM1, BMDATE, BMTIME, RDBMRW
2007     FORMAT(/ 5X, '%%% FILE READ ERROR IN PART 1 IN RDBMAT'
     &         / 5X, 'I/O STATUS = ', I4, 2X, 'NO. OF WORDS = ', I6
     &         / 5X, 'BMDATE = ', I6.6, 2X, 'BMTIME = ', I6.6,
     &2X, 'ROW = ', I3)
         CALL EXIT
      END IF
```

ERROR CHECK #5:

Part 2 of the BMAT file is read by iterating over the number of reduced species. For each iteration, a call to RDMXBM is made. The number of words to read, the starting address of the common block BMFILE, and IOST are passed to RDMXBM. Upon return from RDMXBM, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the I/O status value, the number of words, the time step date and time, the row, and the species number. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C read BMAT part 2 (reduced species list)
      DO 301 ISPC = 1, BMSPRD
         CALL RDMXBM (NWDBM2, AA1(1,1,ISPC), IOST)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2009) IOST, NWDBM2, BMDATE, BMTIME,
     &  RDBMRW, ISPC
2009         FORMAT(/ 5X, '%%% READ ERROR ON PART 2 IN RDBMAT'
    &  / 5X, 'I/O STATUS = ', I4
    &  / 5X, 'NO. OF WORDS = ', I3
    &  / 5X, 'BMDATE = ', I6.6, 5X, 'BMTIME = ', I6.6
    &  / 5X, 'ROW = ', I2, 5X, 'SPECIES NUMBER = ', I2)
CALL EXIT
         END IF
301   CONTINUE
```

SUBROUTINE RDBT

ERROR CHECK #1:

RDBT is called by RDBTRK to read the header record. In RDBT, the header record is read from the BTRK file into the character buffer RECONE. The I/O status value for the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the logical name of the file, the actual name of the file, the I/O status value, and the record length. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read header record
C
      READ(UNITBT, IOSTAT = IOST) RECONE
      IF (IOST .NE. 0) THEN
         INQUIRE (FILE = FLNMBT, NAME = EQNAME)
         WRITE(LUNOUT, 2001) UNITBT, FLNMBT, EQNAME, IOST, RECLEN,
     &           'FIRST RECORD    '
2001     FORMAT(/ 5X, '%%% ERROR READING HEADER RECORD IN RDBT',
     &          / 5X, 'UNITBT = ', I2, 2X, 'FNAME = ', A8
     &          / 5X, 'EQNAME = ', A64
     &          / 5X, 'IOST = ', I4, 2X, 'RECLEN = ', I4, 2X, A16)
         CALL EXIT
       END IF
```

ERROR CHECK #2:

A formatted read of the buffer containing the header information (RECONE) is made to convert the header information into character or numeric type, and the common block HEADBT is loaded. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the logical name of the file, the actual name of the file, the I/O status value, and the record length. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C convert character to mixed character & numeric
C
      READ(RECONE, 1001, IOSTAT = IOST)
     &     CDATBT, CTIMBT, SDATBT, STHRBT, TSTPBT, FRSTBT,
     &     GRDNBT, SWLNBT, SWLTBT, NELNBT, NELTBT, DLONBT,
     &     DLATBT, NCOLBT, NROWBT, NMIFBT, ICNTBT
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 4I4.4)
      IF (IOST .NE. 0) THEN
         INQUIRE (FILE = FLNMBT, NAME = EQNAME)
         WRITE(LUNOUT, 2001) UNITBT, FLNMBT, EQNAME, IOST, RECLEN,
     &           'INTERNAL READ #1'
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The MIF data records are next read by iterating over the number of MIF data records. For each iteration, an MIF record is read into the buffer RECMIF. The I/O status value from the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the logical name of the file, the actual name of the file, the I/O status value, and the record length. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C read the MIF data records
C
      DO 101 IMIF = 1, NMIFBT
        READ(UNITBT, IOSTAT = IOST) RECMIF
        IF (IOST .NE. 0) THEN
           INQUIRE (FILE = FLNMBT, NAME = EQNAME)
           WRITE(CHBUF, 1003) '  MIF - NO. ', IMIF
1003       FORMAT(A12, I4)
           WRITE(LUNOUT, 2001) UNITBT, FLNMBT, EQNAME, IOST,
     & RECLEN, CHBUF
           CALL EXIT
        END IF
```

ERROR CHECK #4:

A formatted read of the buffer containing the MIF information (RECMIF) is made to convert the MIF records to character and numeric data, and the common blocks HEADBT and CHARBT are loaded. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the logical name of the file, the actual name of the file, the I/O status value, and the record length. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
        READ(RECMIF, 1005, IOSTAT = IOST)
     &     MFNMBT(IMIF), CDMFBT(IMIF), CTMFBT(IMIF),
     &     UDMFBT(IMIF), UTMFBT(IMIF)
1005    FORMAT(A12, 4I8.8)
        IF (IOST .NE. 0) THEN
           INQUIRE (FILE = FLNMBT, NAME = EQNAME)
           WRITE(CHBUF, 1003) 'INTERNAL READ #2', IMIF
           WRITE(LUNOUT, 2001) UNITBT, FLNMBT, EQNAME, IOST,
     & RECLEN, CHBUF
           CALL EXIT
        END IF
101   CONTINUE
```

ERROR CHECK #5:

The text records are next read by iterating over the number of text records. For each iteration, a text record is read into the buffer RECTXT. The I/O status value from the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the logical name of the file, the actual name of the file, the I/O status value, and the record length. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C read header text records
C
      DO 201 ITXT = 1, ICNTBT
        READ (UNITBT, IOSTAT = IOST) RECTXT
        IF (IOST .NE. 0) THEN
           INQUIRE (FILE = FLNMBT, NAME = EQNAME)
           WRITE(LUNOUT, 2001) UNITBT, FLNMBT, EQNAME, IOST, RECLEN,
     &         'TEXT RECORDS    '
           CALL EXIT
        END IF
```

ERROR CHECK #6:

A formatted read of the buffer containing the text records (RECTXT) is made to convert the text records to character data, and the common block CHARBT is loaded. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the logical name of the file, the actual name of the file, the I/O status value, and the record length. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
        READ(RECTXT, 1007, IOSTAT = IOST) TEXTBT(ITXT)
1007    FORMAT(A80)
        IF (IOST .NE. 0) THEN
          INQUIRE (FILE = FLNMBT, NAME = EQNAME)
          WRITE(LUNOUT, 2001) UNITBT, FLNMBT, EQNAME, IOST, RECLEN,
     &          'INTERNAL READ   '
          WRITE(LUNOUT, 1009) RECTXT
1009      FORMAT(5X, A80)
          CALL EXIT
        END IF
```

ERROR CHECK #6:

Each row of the backtrack file is read. The I/O status value of the read is stored in the variable IOST. IOST is tested; if IOST is less than zero, then the end-of-file marker is reached. An informational message is written to the log. The backtrack file is closed and control returns to the calling subroutine. If IOST is not equal to zero, then an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the file name, the logical file name, the I/O status value, and the record length. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
C read data record
C
      READ (UNITBT, IOSTAT = IOST) DATA
      RECNO = 1
C
C read body . . . . . . . . . . . . . . . . . . . . . . record*
C
      IF (IOST .LT. 0) GO TO 401
      IF (IOST .NE. 0) THEN
         INQUIRE (FILE = FLNMBT, NAME = EQNAME)
         WRITE(LUNOUT, 2003) UNITBT, FLNMBT, EQNAME, IOST, RECNO
2003     FORMAT(/ 5X, '%%% ERROR READING DATA RECORD IN RDBT',
     &5X, 'CONTROL RETURNED TO RDBTRK'
     &          / 5X, 'UNITBT = ', I2, 2X, 'FNAME = ', A8
     &          / 5X, 'EQNAME = ', A64
     &          / 5X, 'IOST = ', I2, 2X, 'RECNO =  ', I4)
         RETURN
      END IF
   .
   .
   .
401   CONTINUE
      WRITE(LUNOUT, 1011) RECNO, FLNMBT, UNITBT
1011  FORMAT(5X, I6, ' records read on ', A12, ' from unit ', I2 /)
C
      CLOSE (UNIT = UNITBT)
```

SUBROUTINE RDBTRK

ERROR CHECK #1:

The BTRK file is opened by a call to OPBTRK, and the common block HEADBT is loaded. HEADBT contains the header information for the BTRK file. The header information is then tested in the subroutine RDBTRK. If any of the parameters fail the test, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name and the BTRK header information. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C open BTRK file
      CALL OPBTRK
C
C check parameters in headers
      IF ((TSTPBT .NE. TSTPIN) .OR. (GRDNBT .NE. GRDNIN) .OR.
     &    (ABS(SWLNBT - SWLNIN) .GT. 0.001) .OR.
     &    (ABS(SWLTBT - SWLTIN) .GT. 0.001) .OR.
     .
     .
     .
      WRITE(LUNOUT, 2001) TSTPBT, TSTPIN, GRDNBT, GRDNIN,
     &            SWLNBT, SWLNIN, SWLTBT, SWLTIN,
     &            NELNBT, NELNIN, NELTBT, NELTIN,
     &            DLONBT, DLONIN, DLATBT, DLATIN,
     &            NCOLBT, NCOLIN, NROWBT, NROWIN
2001     FORMAT(/ 5X, 'XXX PARAMETER CHECK FAILURE IN RDBTRK'
     &            / 5X, 'TSTPBT = ', I5, 5X, 'TSTPIN = ', I5
     &            / 5X, 'GRDNBT = ', A8, 5X, 'GRDNIN = ', A8
     &            / 5X, 'NELNBT = ', F10.5, 5X, 'NELNIN = ', F10.5
     &            / 5X, 'NELTBT = ', F10.5, 5X, 'NELTIN = ', F10.5
     &            / 5X, 'SWLNBT = ', F10.5, 5X, 'SWLNIN = ', F10.5
     &            / 5X, 'SWLTBT = ', F10.5, 5X, 'SWLTIN = ', F10.5
     &            / 5X, 'DLONBT = ', F10.5, 5X, 'DLONIN = ', F10.5
     &            / 5X, 'DLATBT = ', F10.5, 5X, 'DLATIN = ', F10.5
     &            / 5X, 'NCOLBT = ', I3, 5X, 'NCOLIN = ', I3
     &            / 5X, 'NROWBT = ', I3, 5X, 'NROWIN = ', I3 /)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

The BTRK time step header is read by calling the subroutine RDBT. The number of words to be read, starting address of the common block RTSHBT, and IOST are passed to RDBT. Upon return from the call to RDBT, IOST contains the I/O status of the read operation of the BTRK time step header. IOST is tested; if IOST is less than zero, then an end-of-file marker was reached. Control is returned to the calling subroutine. If IOST is not equal to zero, then an error occurred on the read operation. A message is written to the log to indicate that an error occurred. The message contains the subroutine name, the number of words to read, the value of the I/O status, the time step date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C read BTRK T.S.H.
      CALL RDBT (IWDLN1, DATBT, IOST)
C
      IF (IOST .LT. 0) RETURN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) IWDLN1, IOST, BTDATE, BTTIME
2003     FORMAT(/ 5X, 'XXX T.S.H. READ ERROR IN RDBTRK'
     &            / 5X, 'WDLN1 = ', I2, 2X, 'I/O STATUS = ', I4,
     &2X, 'BTDATE = ', I6.6, 2X, 'BTTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The current model time step time and date are compared with the time step time and date obtained from reading the BTRK time step header. If the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the time step header, and the model date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
      IF (IDATBT .NE. BTDATE .OR. ITIMBT .NE. BTTIME) THEN
         WRITE(LUNOUT, 2005) IDATBT, ITIMBT, BTDATE, BTTIME
2005     FORMAT(/ 5X, '%%% DATES/TIMES DO NOT MATCH IN RDBTRK'
     &          / 5X, 'IDATBT = ', I6.6, 2X, 'ITIMBT = ', I6.6
     &          / 5X, 'BTDATE = ', I6.6, 2X, 'BTTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #4:

The backtrack file records are read by calling subroutine RDBT. The number of words to read, the starting address of the common block BTFILE, and IOST are passed to RDBT. Upon return from RDBT, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred while reading a record from the BTRK file. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the I/O status value, the number of words, the time step date and time, and the row. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read BTRK
      CALL RDBT (IWDLN2, XRU, IOST)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2007) IOST, IWDLN2, BTDATE, BTTIME, RDBTRW
2007     FORMAT(/ 5X, '%%% FILE READ ERROR IN RDBTRK'
     &          / 5X, 'I/O STATUS = ', I4, 2X,
     &          'NUMBER OF WORDS TO BE READ = ', I3
     &          / 5X, 'BTDATE = ', I6.6, 2X, 'BTTIME = ', I6.6,
     &2X, 'ROW = ', I3)
         CALL EXIT
      END IF
```

SUBROUTINE RDCHAR

NONE

SUBROUTINE RDCONC

ERROR CHECK #1:

The CONC file is opened by a call to OPCONC, and the common block HEADCN is loaded. HEADCN contains the header information for the CONC file. The header information is then checked in the subroutine RDCONC. If any of the parameters fail the test, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name and the CONC header information. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C open the CONC file
      CALL OPCONC
C
C check parameters in header
      IF ((TSTPCN .NE. TSTPIN) .OR. (GRDNCN .NE. GRDNIN) .OR.
     &    (ABS(SWLNCN - SWLNIN) .GT. 0.001) .OR.
     &    (ABS(SWLTCN - SWLTIN) .GT. 0.001) .OR.
.
.
.
      WRITE(LUNOUT, 2001) TSTPCN, TSTPIN, GRDNCN, GRDNIN,
     &           SWLNCN, SWLNIN, SWLTCN, SWLTIN,
     &           NELNCN, NELNIN, NELTCN, NELTIN,
     &           DLONCN, DLONIN, DLATCN, DLATIN,
     &           NCOLCN, NCOLIN, NROWCN, NROWIN,
     &           NLEVCN, NLEVIN, NSPCCN, NSPCIN
2001     FORMAT(/ 5X, 'XXX PARAMETER CHECK FAILURE IN RDCONC'
     &          / 5X, 'TSTPCN = ', I5, 5X, 'TSTPIN = ', I5
     &          / 5X, 'GRDNCN = ', A8, 5X, 'GRDNIN = ', A8
     &          / 5X, 'NELNCN = ', F10.5, 5X, 'NELNIN = ', F10.5
     &          / 5X, 'NELTCN = ', F10.5, 5X, 'NELTIN = ', F10.5
     &          / 5X, 'SWLNCN = ', F10.5, 5X, 'SWLNIN = ', F10.5
     &          / 5X, 'SWLTCN = ', F10.5, 5X, 'SWLTIN = ', F10.5
     &          / 5X, 'DLONCN = ', F10.5, 5X, 'DLONIN = ', F10.5
     &          / 5X, 'DLATCN = ', F10.5, 5X, 'DLATIN = ', F10.5
     &          / 5X, 'NCOLCN = ', I3, 5X, 'NCOLIN = ', I3
     &          / 5X, 'NROWCN = ', I3, 5X, 'NROWIN = ', I3
     &          / 5X, 'NLEVCN = ', I3, 5X, 'NLEVIN = ', I3
     &          / 5X, 'NSPCCN = ', I3, 5X, 'NSPCIN = ', I3)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

The CONC time step header is read by calling the subroutine RDFILE. The unit number, the number of words to be read, starting address of the common block RTSHCN, and IOST are passed to RDFILE. Upon return from the call to RDFILE, IOST contains the I/O status of the read operation of the CONC time step header. IOST is tested; if IOST is less than zero, then an end-of-file marker was reached. Control is returned to the calling subroutine. If IOST is not equal to zero, then an error occurred on the read operation. A message is written to the log to indicate that an error occurred. The message contains the subroutine name, the number of words to read, the value of the I/O status, the time step date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C read CONC T.S.H.
      CALL RDFILE (UNITCN, NWDTSH, DATCN, IOST)
C
      IF (IOST .LT. 0) RETURN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) NWDTSH, IOST, CNDATE, CNTIME
2003     FORMAT(/ 5X, '%%% T.S.H. READ ERROR IN RDCONC'
     &          / 5X, 'NO. OF WORDS = ', I2, 2X, 'I/O STATUS = ', I4,
     &2X, 'CNDATE = ', I6.6, 2X, 'CNTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The current model time step time and date are compared with the time step time and date obtained from reading the CONC time step header. If either the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the time step header, and the model date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
      IF (IDATCN .NE. CNDATE .OR. ITIMCN .NE. CNTIME) THEN
         WRITE(LUNOUT, 2005) IDATCN, ITIMCN, CNDATE, CNTIME
2005     FORMAT(/ 5X, '%%% DATES/TIMES DO NOT MATCH IN RDCONC'
     &          / 5X, 'IDATCN = ', I6.6, 2X, 'ITIMCN = ', I6.6
     &          / 5X, 'CNDATE = ', I6.6, 2X, 'CNTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #4:

Each row of the CONC file is read by iterating over the number of species. For each iteration, a call to RDFILE is made. The unit number, the number of words to read, the starting address of the common block CNFILE, and IOST are passed to RDFILE. Upon return from RDFILE, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log indicating that an error has occurred. The message contains the subroutine name, the I/O status value, the number of words, the time step date and time, and the row. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read CONC row
      DO 211 ISPC = 1, NSPECS
         CALL RDFILE (UNITCN, NWDSCN, CNFILE(1,1,ISPC), IOST)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2007) IOST, NWDSCN, CNDATE, CNTIME, RDCNRW
2007        FORMAT(/ 5X, '%%% FILE READ ERROR ON ROW IN RDCONC'
     & / 5X, 'I/O STATUS = ', I4,
     &   2X, 'NO. OF WORDS = ', I6
     & / 5X, 'CNDATE = ', I6.6, 2X, 'CNTIME = ', I6.6,
     &   2X, 'ROW = ', I3)
CALL EXIT
         END IF
211   CONTINUE
```

SUBROUTINE RDFILE

NONE

SUBROUTINE RDHDBM

ERROR CHECK #1:

In RDHDBM, the header record is read from the BMAT file into the buffer RECONE. The I/O status value of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read 1st segment record
C
      READ(UNITBM, IOSTAT = IOST) RECONE
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ISUB, IOST, '%%% FIRST RECORD
2001     FORMAT(/ 5X, '%%% READ ERROR IN RDHDBM'
     &           / 5X, 'SUBFILE NO. ', I2,
     &5X, 'IOSTAT = ', I4, 4X, A24)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

A formatted read of RECONE, the buffer containing the header record, converts the header record into character and numeric data, and loads the record into HEADBM. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read.

A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C convert character to mixed character & numeric and load HEADBM
C
      READ(RECONE, 1001, IOSTAT = IOST)
     &      CDATBM, CTIMBM, SDATBM, STHRBM, TSTPBM, FRSTBM,
     &      GRDNBM, SWLNBM, SWLTBM, NELNBM, NELTBM, DLONBM,
     &      DLATBM, NCOLBM, NROWBM, NLEVBM, NSPCBM, NMIFBM,
     &      ICNTBM
1001  FORMAT(6I8.8, A8, 4F8.3, 2F8.5, 6I4.4)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ISUB, IOST, 'INTERNAL READ, 1ST REC  '
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The species names record is read from the BMAT file and stored in the common block CHARBM. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C read species names record
C
      READ(UNITBM, IOSTAT = IOST) (SPNMBM(ISPC), ISPC = 1, NSPCBM)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ISUB, IOST, '%%% SPECIES NAMES       '
         CALL EXIT
      END IF
```

ERROR CHECK #4:

The index group record is read from the BMAT file into the buffer RECNDX. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read the index group record
C
      READ(UNITBM, IOSTAT = IOST) RECNDX
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ISUB, IOST, '%%% INDEX GROUP          '
         CALL EXIT
      END IF
```

ERROR CHECK #5:

A formatted read of RECNDX, the buffer containing the index group record, converts the index group record to character and numeric data, and loads the record into HEADBM. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C convert character to numeric and load HEADBM
C
      READ(RECNDX, 1003, IOSTAT = IOST)
     &   (BMINDX(ISPC), ISPC = 1, NSPCBM)
1003  FORMAT(<NSPCBM>I4)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ISUB, IOST, 'INTERNAL READ, INDEX REC'
         CALL EXIT
      END IF
```

ERROR CHECK #6:

The layer names record is read from the BMAT file and loaded into CHARBM. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
C read the level names record
C
      READ(UNITBM, IOSTAT = IOST) (LVNMBM(LEV), LEV = 1, NLEVBM)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ISUB, IOST, '%%% LEVEL NAMES          '
         CALL EXIT
      END IF
```

ERROR CHECK #7:

The MIF data records are next read by iterating over the number of MIF data records. For each iteration, an MIF record is read into the buffer RECMIF. The I/O status value from the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #7:

```
C
C read the MIF data records
C
      DO 101 IMIF = 1, NMIFBM
         READ(UNITBM, IOSTAT = IOST) RECMIF
         IF (IOST .NE. 0) THEN
WRITE(CHBUF, 1005) '%%% MIF - NUMBER  ', IMIF
1005       FORMAT(A20, I4)
WRITE(LUNOUT, 2001) ISUB, IOST, CHBUF
CALL EXIT
         END IF
```

ERROR CHECK #8:

A formatted read of RECMIF, the buffer containing the MIF record, converts the MIF record to character and numeric data, and loads the record into HEADBM. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #8:

```
C
        READ(RECMIF, 1007, IOSTAT = IOST)
     &    MFNMBM(IMIF), CDMFBM(IMIF), CTMFBM(IMIF),
     &    UDMFBM(IMIF), UTMFBM(IMIF)
1007    FORMAT(A12, 4I8.8)
        IF (IOST .NE. 0) THEN
WRITE(CHBUF, 1005) 'INTERNAL READ, MIF DATA ', IMIF
WRITE(LUNOUT, 2001) ISUB, IOST, CHBUF
CALL EXIT
        END IF
101   CONTINUE
```

ERROR CHECK #9:

The text records are next read by iterating over the number of text records. For each iteration, a text record is read into the buffer RECTXT. The I/O status value from the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #9:

```
C
C read header text records
C
      DO 201 ITXT = 1, ICNTBM
        READ(UNITBM, IOSTAT = IOST) RECTXT
        IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2001) ISUB, IOST,
     & 'XXX TEXT RECORDS         '
CALL EXIT
        END IF
```

ERROR CHECK #10:

A formatted read of RECTXT, the buffer containing the text records, converts the text records to character data, and loads them into CHARBM. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #10:

```
C
        READ(RECTXT, 1009, IOSTAT = IOST) TEXTBM(ITXT)
1009    FORMAT(A80)
        IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2001) ISUB, IOST,
     & 'INTERNAL READ, TEXT RECS'
WRITE(LUNOUT, 1011) RECTXT
1011        FORMAT(5X, A80)
CALL EXIT
        END IF
201   CONTINUE
```

ERROR CHECK #11:

The subfile header record is read from the BMAT file into the buffer FLCREC. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #11:

```
C
C read subfile header record
C
      READ(UNITBM, IOSTAT = IOST) FLCREC
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ISUB, IOST, 'XXX SUBFILE RECORD      '
         CALL EXIT
      END IF
```

ERROR CHECK #12:

A formatted read of FLCREC, the buffer containing the subfile header record, converts the record to integer data, and loads the record into HEADBM. The I/O status of the read operation is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the subfile number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #12:

```
C
C convert character to numeric and load HEADBM
C
      READ(FLCREC, 1013, IOSTAT = IOST) ISUBFL, NSUBFL,
     &           FRSTSF, LSSTSF
1013  FORMAT(4I4)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) ISUB, IOST, '%%% SUBFILE COUNT        '
         CALL EXIT
      END IF
```

SUBROUTINE RDICON

ERROR CHECK #1:

The ICON file is opened by a call to OPICON, and the common block HEADIC is loaded. HEADIC contains the header information for the ICON file. The header information is then tested in the subroutine RDICON. If any of the parameters fail the test, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name and the ICON header information. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C open the ICON file
      CALL OPICON
C
C check parameters in header
      IF ((TSTPIC .NE. TSTPIN) .OR. (GRDNIC .NE. GRDNIN) .OR.
     &    (ABS(SWLNIC - SWLNIN) .GT. 0.001) .OR.
     &    (ABS(SWLTIC - SWLTIN) .GT. 0.001) .OR.
         WRITE(LUNOUT, 2001) TSTPIC, TSTPIN, GRDNIC, GRDNIN,
     &           SWLNIC, SWLNIN, SWLTIC, SWLTIN,
     &           NELNIC, NELNIN, NELTIC, NELTIN,
     &           DLONIC, DLONIN, DLATIC, DLATIN,
     &           NCOLIC, NCOLIN, NROWIC, NROWIN,
     &           NLEVIC, NLEVIN, NSPCIC, NSPCIN
2001     FORMAT(/ 5X, '%%% PARAMETER CHECK FAILURE IN RDICON'
     &          / 5X, 'TSTPIC = ', I5, 5X, 'TSTPIN = ', I5
     &          / 5X, 'GRDNIC = ', A8, 5X, 'GRDNIN = ', A8
     &          / 5X, 'NELNIC = ', F10.5, 5X, 'NELNIN = ', F10.5
     &          / 5X, 'NELTIC = ', F10.5, 5X, 'NELTIN = ', F10.5
     &          / 5X, 'SWLNIC = ', F10.5, 5X, 'SWLNIN = ', F10.5
     &          / 5X, 'SWLTIC = ', F10.5, 5X, 'SWLTIN = ', F10.5
     &          / 5X, 'DLONIC = ', F10.5, 5X, 'DLONIN = ', F10.5
     &          / 5X, 'DLATIC = ', F10.5, 5X, 'DLATIN = ', F10.5
     &          / 5X, 'NCOLIC = ', I3, 5X, 'NCOLIN = ', I3
     &          / 5X, 'NROWIC = ', I3, 5X, 'NROWIN = ', I3
     &          / 5X, 'NLEVIC = ', I3, 5X, 'NLEVIN = ', I3
     &          / 5X, 'NSPCIC = ', I3, 5X, 'NSPCIN = ', I3)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

The ICON time step header is read by calling the subroutine RDFILE. The unit number, the number of words to be read, starting address of the common block RTSHIC, and IOST are passed to RDFILE. Upon return from the call to RDFILE, IOST contains the I/O status of the read operation of the ICON time step header. IOST is tested; if IOST is less than zero, then an end of file was reached. Control is returned to the calling subroutine. If IOST is not equal to zero, then an error occurred on the read operation. A message is written to the log to indicate that an error occurred. The message contains the subroutine name, the number of words to read, the value of the I/O status, the time step date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C read ICON T.S.H.
      CALL RDFILE (UNITIC, NWDTSH, DATIC, IOST)
C
      IF (IOST .LT. 0) RETURN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) NWDTSH, IOST, ICDATE, ICTIME
2003     FORMAT(/ 5X, 'XXX T.S.H. READ ERROR IN RDICON'
     &          / 5X, 'NO. OF WORDS = ', I2, 2X, 'I/O STATUS = ', I4,
     &2X, 'ICDATE = ', I6.6, 2X, 'ICTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

The current model time step time and date are compared with the time step time and date obtained from reading the ICON time step header. If either the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the time step header, and the model date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
      IF (IDATIC .NE. ICDATE .OR. ITIMIC .NE. ICTIME) THEN
         WRITE(LUNOUT, 2005) IDATIC, ITIMIC, ICDATE, ICTIME
2005     FORMAT(/ 5X, 'XXX DATES/TIMES DO NOT MATCH IN RDICON'
     &          / 5X, 'IDATIC = ', I6.6, 2X, 'ITIMIC = ', I6.6
     &          / 5X, 'ICDATE = ', I6.6, 2X, 'ICTIME = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #4:

Each row of the ICON file is read by iterating over the number of species. For each iteration, a call to RDFILE is made. The unit number, the number of words to read, the starting address of the common block ICFILE, and IOST are passed to RDFILE. Upon return from RDFILE, IOST contains the I/O status of the read operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the I/O status value, the number of words, the time step date and time, and the row. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read ICON row
      DO 211 ISPC = 1, NSPECS
        CALL RDFILE (UNITIC, NWDSIC, ICFILE(1,1,ISPC), IOST)
        IF (IOST .NE. 0) THEN
           WRITE(LUNOUT, 2007) IOST, NWDSIC, ICDATE, ICTIME, IROW
2007       FORMAT(/ 5X, 'XXX FILE READ ERROR ON ROW IN RDICON'
     &/ 5X, 'I/O STATUS = ', I4, 2X, 'NO. OF WORDS = ', I6
     &/ 5X, 'ICDATE = ', I6.6, 2X, 'ICTIME = ', I6.6,
     &  2X, 'ROW = ', I3)
           CALL EXIT
        END IF
211   CONTINUE
```

SUBROUTINE RDMXBM

ERROR CHECK #1:

The BMAT file requires a large amount of disk space and may have been written to several smaller subfiles on different disk packs (since each pack may not individually have had sufficient space to contain the entire file). These subfiles would then be assigned separate logical names in the job's run stream. In subroutine RDMXBM, the subfile number is tested. If the test fails, then the BMAT subfiles are out of order. A call to subroutine DUMPHD is made. The subfile number is the parameter passed to DUMPHD. Upon return from DUMPHD, the program exits by issuing a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C verify subfile
      IF (ISUBFL .NE. ISUB .OR. NSUBFL .NE. NSUB) THEN
         CALL DUMPHD (ISUB)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

A record is read from the BMAT subfile. The I/O status value for the read operation is stored in the variable IOST. IOST is tested; if IOST is less than zero, then the last record was read. An informational message is printed to the log. The BMAT subfile is closed and control is returned to the calling subroutine. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the subfile number, the unit number, the logical name of the file, the actual name of the file, the I/O status value, and the number of records. Control is returned to the calling subroutine.

CODE FOR ERROR CHECK #2:

```
C
C read record
C
      READ (UNITBM, IOSTAT = IOST) BUFF
      RECNO = 1
C
C read subfile body
C
      IF (IOST .LT. 0) GO TO 301
      IF (IOST .NE. 0) THEN
         INQUIRE (FILE = FLNMBM(ISUB), NAME = EQNAME)
         WRITE(LUNOUT, 2001) ISUB, UNITBM, FLNMBM(ISUB), EQNAME,
     &            IOST, RECNO
2001     FORMAT(/ 5X, '%%% ERROR READING RECORD IN RDMXBM',
     &            ' FOR SUBFILE NUMBER', I3,
     &5X, 'CONTROL RETURNED TO RDBMAT'
     &            / 5X, 'UNITBM = ', I2, 2X, 'FNAME = ', A8
     &            / 5X, 'EQNAME = ', A64
     &            / 5X, 'IOST = ', I2, 2X, 'RECNO =  ', I4)
         RETURN
      END IF
   .
   .
   .
301   CONTINUE
      WRITE(LUNOUT, 1001) RECNO, FLNMBM(ISUB), UNITBM
1001  FORMAT(// 5X, I6, ' records read on ', A12,
     &      ' from unit ', I2 /)
      TOTREC = TOTREC + RECNO
C
C close subfile
      INQUIRE (UNIT = UNITBM, NAME = EQNAME)
      CLOSE (UNIT = UNITBM)
      WRITE(6, 1003) ISUB, EQNAME, UNITBM
1003  FORMAT(/ 3X, 'Subfile ', I2, ', ' A64
     &      / 3X, 'closed on unit ', I2)
```

ERROR CHECK #3:

Each record of the last subfile of the BMAT file is read. The I/O status value for the read operation is stored in the variable IOST. IOST is tested; If IOST is less than zero, then the last record was read. An informational message is printed to the log. The BMAT subfile is closed and control is returned to the calling subroutine. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the subfile number, the unit number, the logical name of the file, the actual name of the file, the I/O status value, and the number of records. Control is returned to the calling subroutine.

CODE FOR ERROR CHECK #3:

```
C
C read record
C
      READ (UNITBM, IOSTAT = IOST) BUFF
      RECNO = RECNO + 1
C
C last subfl body . . . . . . . . . . . . . . . . . . . record*
C
      IF (IOST .LT. 0) GO TO 601
      IF (IOST .NE. 0) THEN
         INQUIRE (FILE = FLNMBM(ISUB), NAME = EQNAME)
         WRITE(LUNOUT, 2001) ISUB, UNITBM, FLNMBM(ISUB), EQNAME,
     &           IOST, RECNO
         RETURN
       END IF
  .
  .
  .
C last subfl body  . . . . . . . . . . . . . . . . . . . end
C
601   CONTINUE
      WRITE(LUNOUT, 1001) RECNO, FLNMBM(ISUB), UNITBM
      TOTREC = TOTREC + RECNO
      WRITE(LUNOUT, 1005) NSUB, TOTREC
1005  FORMAT(/ 5X, 'Total number of records read from', I3,
     &        ' subfiles = ', I8)
C
C close subfile
      INQUIRE (UNIT = UNITBM, NAME = EQNAME)
      CLOSE (UNIT = UNITBM)
      WRITE(6, 1003) ISUB, EQNAME, UNITBM
```

SUBROUTINE RDSTAV

ERROR CHECK #1:

A formatted read from the STATE VECTOR file (RESTRT) is made to obtain the STATE VECTOR time step header. This formatted read converts the data into integer data, and loads them into the common block TSHDSV. The I/O status value for the read operation is stored in variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C read STATE VECTOR T.S.H.
       RECNSV = RECNSV + 1
       READ(UNITSV, FMT = 1001, IOSTAT = IOST)
     &      IDATSV, ITIMSV, IELPSV, ISTPSV
1001  FORMAT(1X, I5, 1X, I6, 1X, I8, 1X, I4)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) UNITSV, RECNSV, IOST
2001     FORMAT(/ 5X, 'XXX T.S.H READ ERROR IN RDSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

The current model time step time and date are compared with the time step time and date obtained from reading the STATE VECTOR time step header. If either the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the time step header, and the model date and time. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
      IF (IDATSV .NE. IDATE  .OR.  ITIMSV .NE. ITIME) THEN
         WRITE(LUNOUT, 2003) IDATE, ITIME, IDATSV, ITIMSV
2003     FORMAT(/ 5X, 'XXX DATES/TIMES DO NOT MATCH IN RDSTAV'
     &          / 5X, 'IDATE  = ', I6.6, 2X, 'ITIME  = ', I6.6
     &          / 5X, 'IDATSV = ', I6.6, 2X, 'ITIMSV = ', I6.6)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

A formatted read from the STATE VECTOR file is made to obtain the text pointers record. This formatted read converts the data into integer data, and loads them into the common block TEXTPT. The I/O status value for the read operation is stored in variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C read text pointers
C
      RECNSV = RECNSV + 1
      READ(UNITSV, FMT = 1003, IOSTAT = IOST)
     &     BIGMPT, LILGPT,
     &     BCPSPT, BMPSPT, BTPSPT, CNPSPT, ICPSPT,
     &     RDBCPT, RDBMPT, RDBTPT, RDCNPT, RDICPT,
     &     WRCNPT, WRSVPT
1003  FORMAT(1X, 14(I3, 1X))
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2005) UNITSV, RECNSV, IOST
2005     FORMAT(/ 5X, '%%% TEXTPT READ ERROR IN RDSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #4:

A formatted read from the STATE VECTOR file is made to obtain the row counters record. This formatted read converts the data into integer data, and loads them into the common block ROWSCT. The I/O status value for the read operation is stored in variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C read row counters
C
      RECNSV = RECNSV + 1
      READ(UNITSV, FMT = 1005, IOSTAT = IOST)
     &     BMPSRW, RDBMRW, RDBTRW, RDCNRW
1005  FORMAT(1X, 4(I3, 1X))
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2023) UNITSV, RECNSV, IOST
2023     FORMAT(/ 5X, '%%% ROW COUNTERS READ ERROR IN RDSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #5:

A formatted read from the STATE VECTOR file is made to obtain the scenario time record. This formatted read converts the data into integer data, and loads them into the common block TSTEPS. The I/O status value for the read operation is stored in variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the read. A message is written to the log to indicate that an error has occurred. The error message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C read scenario time
C
      RECNSV = RECNSV + 1
      READ(UNITSV, FMT = 1007, IOSTAT = IOST)
     &     MDDATE, MDTIME, MDELAP, MDSTEP,
     &     BCDATE, BCTIME, BCELAP, BCSTEP,
     &     BMDATE, BMTIME, BMELAP, BMSTEP
1007  FORMAT(1X, 3(I5, 1X, I6, 1X, I8, I4, 1X))
C
      RECNSV = RECNSV + 1
      READ(UNITSV, FMT = 1007, IOSTAT = IOST)
     &     BTDATE, BTTIME, BTELAP, BTSTEP,
     &     CNDATE, CNTIME, CNELAP, CNSTEP,
     &     ICDATE, ICTIME, ICELAP, ICSTEP
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2007) UNITSV, RECNSV, IOST
2007     FORMAT(/ 5X, '%%% SCENARIO TIME READ ERROR IN RDSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
C
C file body . . . . . . . . . . . . . . . . . . . . . . end
```

SUBROUTINE RTPHO

NONE

SUBROUTINE RTSET

NONE

SUBROUTINE RUNMGR

ERROR CHECK #1:

The CONC file is opened, and the file header time and date information is tested against the STATE VECTOR file (RESTRT) header time and date. If either the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the CONC file time step header, and the date and time from the STATE VECTOR file time step header.

CODE FOR ERROR CHECK #1:

```
C
C open CONC file and check file header with STATE VECTOR file header
      CALL OPCONC
      IF (CDATCN .NE. CDATSV .OR. CTIMCN .NE. CTIMSV) THEN
         WRITE(LUNOUT, 2001) CDATCN, CTIMCN, CDATSV, CTIMSV
2001     FORMAT(// 22X, '!!! WARNING IN RUN MANAGER !!!'
     &          / 36X, 'CONC FILE CREATION DATE/TIME: ', I6, 2X, I6
     &          / 3X, 'DOES NOT MATCH CREATION DATE/',
     &          'TIME ON STATE VECTOR FILE HEADER: ', I6, 2X, I6 //)
      END IF
```

ERROR CHECK #2:

The BCON file is opened. The CONC file header time and date information is tested against the BCON file header time and date. If either the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the CONC file time step header, and the date and time from the BCON file time step header.

CODE FOR ERROR CHECK #2:

```
C
C open BCON file and check file header
      CALL OPBCON
      IF (CDATBC .NE. CDBCCN .OR. CTIMBC .NE. CTBCCN) THEN
         WRITE(LUNOUT, 2003) 'BCON', CDATBC, CTIMBC, CDBCCN, CTBCCN
2003     FORMAT(// 22X, '!!! WARNING IN RUN MANAGER !!!'
     &          / 29X, A4, ' FILE CREATION DATE/TIME: ', I6, 2X, I6
     &          /  4X, 'DOES NOT MATCH CREATION DATE/',
     &          'TIME ON CONC FILE HEADER: ', I6, 2X, I6 //)
      END IF
```

ERROR CHECK #3:

The BTRK file is opened. The CONC file header time and date information is tested against the BTRK file header time and date. If either the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the CONC file time step header, and the date and time from the BTRK file time step header.

CODE FOR ERROR CHECK #3:

```
C
C open BTRK file and check file header
      CALL OPBTRK
      IF (CDATBT .NE. CDBTCN .OR. CTIMBT .NE. CTBTCN) THEN
         WRITE(LUNOUT, 2003) 'BTRK', CDATBT, CTIMBT, CDBTCN, CTBTCN
      END IF
```

ERROR CHECK #4:

The BMAT file is opened. The CONC file header time and date information is tested against the BMAT file header time and date. If either the time or date do not match, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, date and time from the CONC file time step header, and the date and time from the BMAT file time step header.

CODE FOR ERROR CHECK #4:

```
C
C open BMAT file and check file header
      CALL OPBMAT
      IF (CDATBM .NE. CDBMCN .OR. CTIMBM .NE. CTBMCN) THEN
         WRITE(LUNOUT, 2003) 'BMAT', CDATBM, CTIMBM, CDBMCN, CTBMCN
      END IF
```

ERROR CHECK #5:

The CONC file time step header is read by calling the subroutine RDCONC. The parameter IOST is passed to RDCONC. Upon return from subroutine RDCONC, IOST contains the I/O status of the read of the CONC time step header. IOST is tested; if IOST is less than zero, then an end-of-file marker is reached while reading the CONC file. A message is written to the log stating that an end-of-file marker is reached, the I/O status value, and the unit number. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
C read CONC T.S.H
      CALL RDCONC (IOST)
      IF (IOST .LT. 0) THEN
         WRITE(LUNOUT, 2005) IOST, UNITCN
2005     FORMAT(/ 5X, 'XXX EOF REACHED ON CONC FILE'
     &          / 5X, 'I/O STATUS = ', I8
     &          / 5X, 'UNIT       = ', I3 )
         CALL EXIT
      END IF
```

ERROR CHECK #6:

Each row of the CONC file is copied to the BGICCN file by calling subroutine RDCONC. The parameter IOST is passed to RDCONC. Upon return from subroutine RDCONC, IOST contains the I/O status of the read of a row of the CONC file. IOST is tested; if IOST is less than zero, then an end-of-file marker is reached while reading the CONC file. A message is written to the log stating that an end-of-file marker is reached, the I/O status value, and the unit number. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
C copy CONC rows to ICCN file
      DO 101 IROW = 1, NROWIN
         CALL RDCONC (IOST)
         IF (IOST .LT. 0) THEN
WRITE(LUNOUT, 2005) IOST, UNITCN
CALL EXIT
         END IF
```

ERROR CHECK #7:

A test is made to determine whether the run is a full run or a restart run. The variable RUNMODE is tested. If RUNMODE does not equal 'START' or 'RESTART', then RUNMODE is not valid. A message is written to the log to indicate that the mode is incorrect and the value of RUNMODE is written to the log. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #7:

```
C
         IF (RUNMODE .EQ. START) THEN
C
C start run . . . . . . . . . . . . . . . . . . . . . . . . o
C
 .
 .
 .
         ELSE IF (RUNMODE .EQ. RESTART) THEN
C
C restart run  . . . . . . . . . . . . . . . . . . . . . . o
C
 .
 .
 .
         ELSE
C
C error  . . . . . . . . . . . . . . . . . . . . . . . . . o
C
WRITE(LUNOUT, 2007) RUNMODE
2007        FORMAT(// 5X, '%%% ERROR SETTING RUN MODE IN',
        &' RUN MANAGER; MODE WAS SET TO: ', A12 //)
CALL EXIT
         END IF
```

ERROR CHECK #8:

For each model time step, a call to subroutine BIGGAM is made. The parameter IEOFBG is passed to BIGGAM. In subroutine BIGGAM, each file (ICON, BCON, BTRK, AND BMAT) is read. The time step header is read from each file along with every row for the model step. If an end-of-file marker is reached while reading any one of the above files, or if an error was reached while reading the files, then the parameter IEOFBG is set to a specified number. Upon return from BIGGAM, IEOFBG is tested. If it is not equal to zero, then an end-of-file marker or error was encountered while reading the input files. A message is written to the log indicating in which file the error or end-of-file marker was reached, along with the model time step and date. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #8:

```
C
C run body . . . . . . . . . . . . . . . . . . . . . time step*
        DO 301 JSTEP = 1, NSTEPS
C
C read MODEL T.S.H.
          CALL BIGGAM (IEOFBG)
C
          IF (IEOFBG .EQ. 1) THEN
 WRITE(LUNOUT, 2009) 'ICON', IDATMD, ITIMMD
2009        FORMAT(/ 5X, '%%% EOF ON ', A4, ' FILE AT MODEL DATE:',
      & I6, 2X, I6)
 CALL EXIT
          ELSE IF (IEOFBG .EQ. 2) THEN
 WRITE(LUNOUT, 2009) 'BCON', IDATMD, ITIMMD
 CALL EXIT
          ELSE IF (IEOFBG .EQ. 3) THEN
 WRITE(LUNOUT, 2009) 'BTRK', IDATMD, ITIMMD
 CALL EXIT
          ELSE IF (IEOFBG .EQ. 7) THEN
 WRITE(LUNOUT, 2009) 'BMAT', IDATMD, ITIMMD
 CALL EXIT
          ELSE IF (IEOFBG .EQ. 4) THEN
 WRITE(LUNOUT, 2011) 'ICON', IDATMD, ITIMMD
2011        FORMAT(/ 5X, '%%% ERROR READING ', A4,
      &     ' FILE AT MODEL DATE:',
      & I6, 2X, I6)
 CALL EXIT
          ELSE IF (IEOFBG .EQ. 5) THEN
 WRITE(LUNOUT, 2011) 'BCON', IDATMD, ITIMMD
          ELSE IF (IEOFBG .EQ. 6) THEN
 WRITE(LUNOUT, 2011) 'BTRK', IDATMD, ITIMMD
          END IF
```

SUBROUTINE TIMER

NONE

SUBROUTINE WRCHAR

NONE

SUBROUTINE WRCONC

ERROR CHECK #1:

The CONC time step header is written by calling the subroutine WRFILE. The unit number, the number of words to be written, starting address of the common block RTSHCN, and IOST are passed to WRFILE. Upon return from the call to WRFILE, IOST contains the I/O status of the write operation of the CONC time step header. IOST is tested. If IOST is less than zero, then an end-of-file marker was reached. Control is returned to the calling subroutine. If IOST is not equal to zero, then an error occurred on the write operation. A message is written to the log to indicate that an error occurred. The message contains the subroutine name, the unit number, the I/O status value, and the number of words to write. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C write CONC T.S.H.
C
      CALL WRFILE (UNITCN, IWLTSH, DATCN, IOST)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) UNITCN, IOST, IWLTSH
2001     FORMAT(// 5X, '%%% ERROR WRITING TSH IN WRCONC'
     &          / 5X, 'UNIT NUMBER = ', I2, 2X, 'I/O STATUS = ', I4,
     & 2X, 'NO. OF WORDS = ', I4 /)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

Each row of the CONC file is written by iterating over the number of species. For each iteration, a call to WRFILE is made. The unit number, the number of words to read, the starting address of the common block CNFILE, and IOST are passed to WRFILE. Upon return from WRFILE, IOST contains the I/O status of the write operation. If IOST is not equal to zero, then an error has occurred. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the I/O status value, the number of words, the time step date and time, and the row. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C write CONC row
      DO 301 ISPC = 1, NSPECS
        CALL WRFILE (UNITCN, IWDLN2, CNFILE(1,1,ISPC), IOST)
        IF (IOST .NE. 0) THEN
           WRITE(LUNOUT, 2003) UNITCN, IOST, IWDLN2
2003       FORMAT(// 5X, '%%% ERROR WRITING ROW IN WRCONC'
     & / 5X, 'UNIT NUMBER = ', I2,
     &   2X, 'I/O STATUS = ', I4,
     &   2X, 'NO. OF WORDS = ', I4 /)
           CALL EXIT
        END IF
301   CONTINUE
```

SUBROUTINE WRFILE

NONE

SUBROUTINE WRSTAV

ERROR CHECK #1:

The STATE VECTOR file is opened. The I/O status value of the open statement is stored in variable IOST. IOST is tested; if IOST does not equal zero, an error occurred on the open statement. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #1:

```
C
C open STATE VECTOR file, formatted, read/write access
C
      UNITSV = JUNIT()
      OPEN (UNITSV,
     &      FILE = FLNMSV,
     &      ACCESS = 'SEQUENTIAL',
     &      STATUS = 'UNKNOWN',
     &      IOSTAT = IOST)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2001) UNITSV, IOST
2001     FORMAT(/ 5X, '%%% SV FILE OPEN ERROR IN WRSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #2:

A formatted write of the common blocks CHARSV and HEADSV writes the first header segment record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #2:

```
C
C write STATE VECTOR header segment 1
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1001, IOSTAT = IOST)
     &      CDATSV, CTIMSV, SDATSV, STHRSV, TSTPSV,
     &      FRSTSV, GRDNSV, SWLNSV, SWLTSV, NELNSV
1001  FORMAT(1X,  2(I6, 1X), I5, 1X, I2, 1X, 2(I8, 1X),
     &        A8, 1X, 3(F8.3, 1X))
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2003) UNITSV, RECNSV, IOST
2003     FORMAT(/ 5X, '%%% HEADER WRITE ERROR IN WRSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #3:

A formatted write of the common block HEADSV writes the second header segment record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #3:

```
C
C write STATE VECTOR header segment 2
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1003, IOSTAT = IOST)
     &      NELTSV, DLONSV, DLATSV, NCOLSV,
     &      NROWSV, NLEVSV, NSPCSV, ICNTSV
1003  FORMAT(1X, F8.3, 2(1X, F8.5), 5(1X, I4))
      IF (IOST.NE.0) THEN
         WRITE(LUNOUT, 2003) UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #4:

A formatted write of the common block CHARSV writes the first segment of the species names record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #4:

```
C
C write the species names records
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1005, IOSTAT = IOST)
     &     (SPNMSV(ISPC), ISPC = 1, 15)
1005  FORMAT(1X, 15(A4, 1X))
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2005) UNITSV, RECNSV, IOST
2005     FORMAT(/ 5X, '%%% SPECIES NAME WRITE ERROR IN WRSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #5:

A formatted write of the common block CHARSV writes the second segment of the species names record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. The second species names record is written. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #5:

```
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1005, IOSTAT = IOST)
     &     (SPNMSV(ISPC), ISPC = 16, NSPCSV)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2005)  UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #6:

A formatted write of the common block CHARSV writes the levels names record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #6:

```
C
C write the level names record
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1005, IOSTAT = IOST)
     &     (LVNMSV(ILEV), ILEV = 1, NLEVSV)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2007) UNITSV, RECNSV, IOST
2007     FORMAT(/ 5X, 'XXX LEVEL NAMES WRITE ERROR IN WRSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #7:

The header text records are written by iterating over the number of text records. For each iteration, a formatted write of the common block CHARSV writes a text record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #7:

```
C
C write header text records
C
      DO 109 ITXT = 1, ICNTSV
         RECNSV = RECNSV + 1
         WRITE(UNITSV, FMT = 1007, IOSTAT = IOST) TEXTSV(ITXT)
1007     FORMAT(1X, A80)
         IF (IOST .NE. 0) THEN
WRITE(LUNOUT, 2009) UNITSV, RECNSV, IOST
2009        FORMAT(/ 5X, 'XXX SV HEADER WRITE ERROR IN WRSTAV'
     & / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     & / 5X, 'I/O STATUS = ', I4)
CALL EXIT
         END IF
109   CONTINUE
```

ERROR CHECK #8:

A formatted write of the common block HEADIN writes the first header segment record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #8:

```
C
C write HEADIN header record segment 1
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1001, IOSTAT = IOST)
     &      CDATIN, CTIMIN, SDATIN, STHRIN, TSTPIN, FRSTIN,
     &      GRDNIN, SWLNIN, SWLTIN, NELNIN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2011) UNITSV, RECNSV, IOST
2011     FORMAT(/ 5X, '%%% FILE WRITE ERROR IN WRSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #9:

A formatted write of the common block HEADIN writes the second header segment record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #9:

```
C
C write HEADIN header record segment 2
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1003, IOSTAT = IOST)
     &      NELTIN, DLONIN, DLATIN, NCOLIN, NROWIN,
     &      NLEVIN, NSPCIN, ICNTIN
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2011) UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #10:

A formatted write of the common block CHEMIN writes the header record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #10:

```
C
C write CHEMIN header record
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1013, IOSTAT = IOST)
     &      ATS, GTS, UFRAX, BFRAX, FACTOR,
     &      DIVP, DIVQ, NCOUT
1013  FORMAT(1X, 2(F8.2, 1X), 5(F8.5, 1X), I5)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2013) UNITSV, RECNSV, IOST
2013     FORMAT(/ 5X, '%%% CHEMIN INTERNAL WRITE ERROR IN WRSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #11:

A formatted write of the common block CHEMIN writes the species record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #11:

```
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1015, IOSTAT = IOST)
     &    (ISPEC(ISPC), ISPC = 1, NCOUT), ULIM, BLIM, FNOLIM
1015  FORMAT(1X, <NCOUT>(I4.3, 1X), 3(E10.3, 1X) )
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2013) UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #12:

A formatted write of the common block NDXSPC writes the first segment record of the species ordering header record (NXSPBM) to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #12:

```
C
C write species ordering header record
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPBM(ISPC), ISPC = 1, 15)
1019  FORMAT (1X, 15(I3, 1X))
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2015) 'BM-1', UNITSV, RECNSV, IOST
2015     FORMAT(/ 5X, '*** FILE WRITE ERROR IN WRSTAV: NXSP', A4
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #13:

A formatted write of the common block NDXSPC writes the second segment record of the species ordering header record (NXSPBM) to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #13:

```
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPBM(ISPC), ISPC = 16, NSPECS)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2015) 'BM-2', UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #14:

A formatted write of the common block NDXSPC writes the first segment record of the species ordering header record (NXSPBC) to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #14:

```
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPBC(ISPC), ISPC = 1, 15)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2015) 'BC-1', UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #15:

A formatted write of the common block NDXSPC writes the second segment record of the species ordering header record (NXSPBC) to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #15:

```
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPBC(ISPC), ISPC = 16, NSPECS)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2015) 'BC-2', UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #16:

A formatted write of the common block NDXSPC writes the first segment record of the species ordering header record (NXSPIC) to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #16:

```
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPIC(ISPC), ISPC = 1, 15)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2015) 'IC-1', UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #17:

A formatted write of the common block NDXSPC writes the second segment record of the species ordering header record (NXSPIC) to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #17:

```
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1019, IOSTAT = IOST)
     &     (NXSPIC(ISPC), ISPC = 16, NSPECS)
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2015) 'IC-2', UNITSV, RECNSV, IOST
         CALL EXIT
      END IF
```

ERROR CHECK #18:

A formatted write of the common block TSHDSV writes the time step header record of the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #18:

```
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1031, IOSTAT = IOST)
     &      IDATSV, ITIMSV, IELPSV, ISTPSV
1031  FORMAT(1X, I5, 1X, I6, 1X, I8, 1X, I4)
C
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2017) UNITSV, RECNSV, IOST
2017     FORMAT(/ 5X, '%%% T.S.H WRITE ERROR IN WRSTAV'
     &           / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &           / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #19:

A formatted write of the common block TEXTPT writes the text pointers record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #19:

```
C
C write text pointers
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1033, IOSTAT = IOST)
     &      BIGMPT, LILGPT,
     &      BCPSPT, BMPSPT, BTPSPT, CNPSPT, ICPSPT,
     &      RDBCPT, RDBMPT, RDBTPT, RDCNPT, RDICPT,
     &      WRCNPT, WRSVPT
1033  FORMAT(1X, 14(I3, 1X))
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2019) UNITSV, RECNSV, IOST
2019     FORMAT(/ 5X, '%%% TEXTPT WRITE ERROR IN WRSTAV'
     &           / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &           / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #20:

A formatted write of the common block ROWSCT writes the row counters record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #20:

```
C
C write row counters
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1035, IOSTAT = IOST)
     &        BMPSRW, RDBMRW, RDBTRW, RDCNRW
1035  FORMAT(1X, 4(I3, 1X))
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2021) UNITSV, RECNSV, IOST
2021     FORMAT(/ 5X, '%%% ROW COUNTERS WRITE ERROR IN WRSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

ERROR CHECK #21:

A formatted write of the common block TSTEPS writes the scenario time record to the STATE VECTOR file. The I/O status of the formatted write statement is stored in the variable IOST. IOST is tested; if IOST does not equal zero, an error has occurred on the write operation. A message is written to the log to indicate that an error has occurred. The message contains the subroutine name, the unit number, the record number, and the I/O status value. The program exits by a call to the system subroutine EXIT.

CODE FOR ERROR CHECK #21:

```
C
C write scenario time
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1037, IOSTAT = IOST)
     &        MDDATE, MDTIME, MDELAP, MDSTEP,
     &        BCDATE, BCTIME, BCELAP, BCSTEP,
     &        BMDATE, BMTIME, BMELAP, BMSTEP
1037  FORMAT(1X, 3(I5, 1X, I6, 1X, I8, I4, 1X))
C
      RECNSV = RECNSV + 1
      WRITE(UNITSV, FMT = 1037, IOSTAT = IOST)
     &        BTDATE, BTTIME, BTELAP, BTSTEP,
     &        CNDATE, CNTIME, CNELAP, CNSTEP,
     &        ICDATE, ICTIME, ICELAP, ICSTEP
      IF (IOST .NE. 0) THEN
         WRITE(LUNOUT, 2023) UNITSV, RECNSV, IOST
2023     FORMAT(/ 5X, '%%% SCENARIO TIME WRITE ERROR IN WRSTAV'
     &          / 5X, 'UNIT NUMBER = ', I2, 5X, 'RECORD = ', I4
     &          / 5X, 'I/O STATUS = ', I4)
         CALL EXIT
      END IF
```

This page is intentionally left blank.

INDEX

Chemical species
mapping, 39

Error checking in
ADATE, F-2
ASORT, F-2
BCPRCS, F-2
BIGGAM, F-3
BMPRCS, F-5
BTPRCS, F-6
CELLM, F-6
CLOCK1, F-7
CLOCK2, F-7
CNPRCS, F-7
CPUTIM, F-7
DATTIM, F-7
DUMPHD, F-9
FSKIP1, F-10
GTILDE, F-13
HSTEP, F-13
ICPRCS, F-13
INDEX1, F-14
INIRUN, F-14
IOCL, F-14
JFILE2, F-14
JFILE5, F-15
JFILE6, F-16
JULIAN, F-17
JUNIT, F-17
LILGAM, F-18
NEWICS, F-19
OPBCON, F-23
OPBMAT, F-25
OPBTRK, F-25
OPCONC, F-26
OPICON, F-28
OPSTAV, F-30
OPWRCN, F-35
ORSPBC, F-39
ORSPBM, F-40
ORSPIC, F-41
POBCON, F-42
POBTRK, F-43
POCONC, F-44
POICON, F-45
POMXBM, F-47
POSTAV, F-48
PQ1, F-49
PQCOEF, F-50
PRGSMY, F-50
RATED, F-51
RDBCON, F-51
RDBMAT, F-54
RDBT, F-57
RDBTRK, F-61
RDCHAR, F-63
RDCONC, F-64
RDFILE, F-66
RDHDBM, F-66
RDICON, F-72
RDMXBM, F-74
RDSTAV, F-76
RTPHO, F-79
RTSET, F-79
RUNMGR, F-79
TIMER, F-83
WRCHAR, F-83
WRCONC, F-83
WRFILE, F-85
WRSTAV, F-85

Include files (.EXT)
ADVSFL, E-3
BCFILE, E-3
BGBCFL, E-3
BGBTFL, E-4
BGICCN, E-4
BMCOEF, E-5
BMFILE, E-6
BTFILE, E-7
CHEMIN, E-7
CHEMSW, E-8
CNFILE, E-8
CONFAC, E-8
DIMENS, E-2
ERRG, E-9
FLNAMS, E-9
GTCOEF, E-10
HDFMTS, E-10
HDSTAV, E-11
HEADBC, E-12
HEADBM, E-13
HEADBT, E-14
HEADCN, E-15
HEADIC, E-16
HEADIN, E-17
HSTEPS, E-18
ICFILE, E-18
LGBMFL, E-19
LILGSP, E-20
LUNITS, E-20
LVNAME, E-21
NDXPC, E-21
NROOTS, E-21
REGION, E-2
RKLEVS, E-22
ROWSCT, E-22
RTCONS, E-23
RTSHBC, E-23
RTSHBM, E-24
RTSHBT, E-24
RTSHCN, E-25
RTSHIC, E-25
RUNTMS, E-26
SPNAME, E-26
STOPFL, E-26
SUBID, E-27
TEXTPT, E-28
TILDE, E-28
TSHDBC, E-29
TSHDBM, E-29
TSHDBT, E-29
TSHDCN, E-30
TSHDIC, E-30
TSHDMD, E-30
TSHDSV, E-31
TSTEPS, E-31
UNITIO, E-32
ZADVSL, E-32

Input files
BCON, 27
BMAT, 34
BTRK, 48
ICON, 54
NEWICON, 60
PROG, 68
RESTRT, 69

Output file
CONC, 90

Procedures, 3, 69
Processes, 3, 69
Program inversion, A-4

Restarting the core model
RESTRT, 70
ROM
chemical species in, 8
CPU time on a VAX 8650, 122
episode, 3
horizontal grid resolution, 3
IBM 3090 clock time, 2
IBM 3090 CPU time, 14
modeling domains, 4
principal components, 10
start time of the Core Model, 13
time zone, 3
vertical layers, 5
Run manager, 16

Start time, 13
Starting the core model
RESTRT, 70
State vectors, A-4
Subprograms
ADATE, 23
ASORT, 23
BCPRCS, 18
BLKMOD, 24
BMPRCS, 20
BTPRCS, 19
CELLM, 23
CLOCK1, 23
CLOCK2, 23
CNPRCS, 21
CPUTIM, 24
DATTIM, 23
DUMPHD, 21
FSKIP1, 23
GTILDE, 21
HSTEP, 21
ICPRCS, 17
INDEX1, 23
INIRUN, 16
IOCL, 23
JFILE2, 23
JFILE5, 23
JFILE6, 23
JULIAN, 23
JUNIT, 23
NEWICS, 24
OPBCON, 18
OPBMAT, 20
OPBTRK, 19
OPCONC, 22
OPICON, 17
OPSTAV, 22
OPWRCN, 21
ORSPBC, 19
ORSPBM, 21
ORSPIC, 18
POBCON, 19
POBTRK, 19
POCONC, 22
POICON, 18
POMXBM, 21
POSTAV, 22
PQ1, 21
PQCOEF, 21
PRGSMY, 24
RATED, 21
RDBCON, 18
RDBMAT, 20
RDBT, 19
RDCHAR, 23
RDCONC, 22
RDFILE, 24
RDHDBM, 20
RDICON, 17
RDMXBM, 20
RDSTAV, 22
RTBTRK, 19
RTPHO, 21
RTSET, 21
TIMER, 24
WRCHAR, 24
WRCONC, 21
WRFILE, 24
WRSTAV, 22

Variable-state subroutines, 3, 69, A-4
VAX to IBM translation, 14