NISTIR 4343

THE CONSOLIDATED COMPARTMENT FIRE MODEL (CCFM) COMPUTER CODE APPLICATION CCFM.VENTS - PART II: SOFTWARE REFERENCE GUIDE

Glenn P. Forney
Leonard Y. Cooper

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Center for Fire Research
Gaithersburg, MD 20899

Sponsored in part by:
Naval Sea Systems Command
Department of the Navy
Washington, DC 20362

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

NIST

NISTIR 4343

THE CONSOLIDATED COMPARTMENT FIRE MODEL (CCFM) COMPUTER CODE APPLICATION CCFM. VENTS - PART II: SOFTWARE REFERENCE GUIDE

Glenn P. Forney
Leonard Y. Cooper

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology
Center for Fire Research
Gaithersburg, MD 20899

Sponsored in part by:
Naval Sea Systems Command
Department of the Navy
Washington, DC 20362

July 1990

U.S. DEPARTMENT OF COMMERCE
Robert A. Mosbacher, Secretary
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
John W. Lyons, Director

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

The Consolidated Compartment Fire Model (CCFM) Computer Code Application CCFM.VENTS - Part IV: Software Reference Guide

Glenn P. Forney and Leonard Y. Cooper

ABSTRACT

A project was carried out at The National Institute of Standards and Technology (NIST) to study the feasibility of developing a new-generation, multi-room, compartment fire model computer code, called the Consolidated Compartment Fire Model (CCFM) computer code. The idea was that such a code would consolidate past progress in zone-type compartment fire modeling, and allow readily for integration of future advances with the greatest possible flexibility. Desired features of the CCFM would include: comprehensive documentation, user-friendliness, significant modularity, numerical robustness, and versatility in the sense that the code would provide a capability of analyzing a particular compartment fire problem by using any one of a range of physical-phenomena-modeling sophistication, from the most basic to the most comprehensive. The project led to the development of a prototype multi-room CCFM product called CCFM.VENTS. CCFM.VENTS involves a model formulation and code structure that allows for the required future CCFM growth flexibility. It has a relatively sophisticated and very general room-to-room forced and unforced vent flow capability. Finally, the CCM.VENTS code uses the simplest possible, point-source-plume, smoke-filling fire physics in the rooms-of-fire-origin and a very simple heat transfer calculation there and in other spaces.

This is Part II of a four-part report which documents CCFM.VENTS. The main objective of this Part II document is to document the design and the underlying structure of the CCFM.VENTS computer software. It serves as a guide for those persons interested in extending, modifying and if necessary correcting CCFM.VENTS at a later date.

The other three parts of this report are: Part I: Physical Basis; Part III: Catalog of Algorithms and Subroutines; Part IV: Users' Reference Guide.

Keywords: **building fires; compartment fires; computer models; fire models; mathematical models; vents; zone models**

1. Introduction and Overview

The purpose of this document is to give the reader insight into the underlying structure and design of CCFM.VENTS. This document is intended for those users of CCFM.VENTS who wish to

- extend its capabilities by installing additional physical models, modifying input requirements, output display and/or modifying data structures

- understand the inner workings and structure of CCFM.VENTS.

In part I [10], the equations CCFM.VENTS uses to model fire phenomena are described. In part III [11], the implementation of these equations into computational algorithms is presented . Part IV [12] is a guide to using the software. This part (part II) explains how the physical algorithms are linked together to form CCFM.VENTS. It also serves as a guide so that someone other than the original developers can extend, modify, and if necessary correct CCFM.VENTS.

The framework or structure of CCFM.VENTS has two aspects: the algorithms or **program structure** and the **data structures**. The algorithms are the plans for what calculations to perform and how to implement them. The data structures are the plans for storing the quantities that are required by the algorithms. Program algorithms and data structures must be designed in concert in order to achieve a practical implementation. Program attributes such as modularity, portability and user friendliness are of little value if the desired computational results are not achieved in a timely fashion.

The rest of this section gives an overview of the strategy used to program CCFM.VENTS. Section 2 discusses the structure of CCFM.VENTS. Section 3 explains how to modify/add program and data structure elements, how to install CCFM.VENTS on an IBM PC compatible computer and what changes are required to move CCFM.VENTS to another computing environment. The appendices document CCFM.VENTS' program and data structures. Appendices A and B are glossaries for the program and data structures respectively. Appendix C lists the data file used by CCFM.VENTS to define its menus and also the default fire scenario. This data file is called a menu definition file and is discussed in Section 2.2.2.

To obtain a quick overview of CCFM.VENTS examine the figures, algorithms and tables of this document first. The figures illustrate the program and data structures, the algorithms detail the calculations and the tables detail the key data structures for each portion of CCFM.VENTS.

1.1 CCFM.VENTS Design Philosophy

The primary objective in the design of CCFM.VENTS was to translate the physical or modeling equations into **robust, efficient** simulation software. A secondary objective was to design the software modules of CCFM.VENTS so that they can be used in successors to this model. The catalog of algorithms documented in Part III [11], the INPUT module and utilities such as the Memory Management Routines and the Character handling routines were written with this feature in mind.

CCFM.VENTS was designed to be **portable,** in order to improve its robustness . A portable program requires few software changes to install it on other computers. Any one computer/Fortran compiler will not find all of a program's flaws and defects. Hidden assumptions built into a program's design can be un-covered by porting it to several computers. In order to achieve portability, a programming style had to be developed. Simply stated it was decided to strictly follow the ANSI-FORTRAN 77 standard as defined by [1].

Software Modules in CCFM.VENTS were designed where possible to be **re-usable**. Plume, natural vent and forced vent calculations are designed to be independent of the internal data structures of CCFM.VENTS. All required input values and calculated output values are passed through the subroutine argument lists. So they can be pulled out of CCFM.VENTS and used in other fire models. The storage framework of CCFM.VENTS is not used within the low-level physical routines, in other words these routines do not access CCFM.VENTS' common blocks.

Another feature important in the design of CCFM.VENTS is its **maintainability**. UPDATE, a software tool on the CYBER 855/NOS was used to manage the development of CCFM.VENTS. Similar tools exist on other computer systems. The purpose of this type of tool is to keep track of software changes. Typically only differences are saved. Many versions of a software program can be stored in one file by saving the differences between the current and the original version. This usually takes less space than storing the complete program. In a large program it is very easy (too easy) to make changes and then to forget where these changes were made. Using tools such as this allows software to be developed in a team environment.

1.2 CCFM.VENTS Programming Style Guidelines

In order to achieve the program design goals set forth in the previous section, a set of programming guide-lines were developed. The guidelines are not arbitrary but serve a definite purpose as detailed below.

1.2.1 Portability

Portability Guideline: Purpose of guideline: To simplify the installation of CCFM.VENTS on other computers.

- Strictly Follow FORTRAN 77 ANSI standard

Requirements not covered by the standard or parts of a program that may need to be changed when it is ported to another computer are:

- file naming convention   the allowed characters and length of file names vary from computer to computer
- screen graphics
- floating point environment

CCFM.VENTS is designed to run on a number of different computers with minimal coding differences between the versions. It has been ported to micro-computers such as the Apple Macintosh, the IBM PC and compatibles; mainframes such as the Cyber 855 and UNIX work stations such as the Silicon Graphics Personal Iris, the Convex C120, the Sun Sparcstation 1, the Vax 8650 and the IBM Risc 6000 Powerstation. Test cases run on the various computers listed above produce results consistent with the solver error tolerances.

There are only 5 to 10 lines of code that are different between the Cyber 855, Apple Macintosh and IBM PC versions. There are no differences in the various UNIX work station versions of CCFM.VENTS. Most of the coding changes required to get CCFM.VENTS to run on a different computer are in the input or output areas that are not addressed by the ANSI-77 Fortran standard. For example the default name for the terminal input/output files for the Cyber is 'INPUT' and 'OUTPUT' but is 'C:' for the Concurrent 3252.

The floating point characteristic of most interest is the unit rounding or machine unit which is defined to be the smallest positive floating point number which can be added to 1 to produce a machine number different from 1. This is not the same as the smallest positive floating point number on the computer. This machine dependent parameter depends on the length of the fractional portion (mantissa) of the machine representation for a real number. Getting machine dependent parameters correct is critical to insuring that a scientific code runs properly in different computing environments.

The hard part then is designing a code to minimize the number of required changes. For machine dependent parameters a popular approach for FORTRAN is to put the parameters for all commercial machines into data statements in function subprograms with the names I1MACH, for integer parameters, R1MACH, for real parameters, and D1MACH, for double precision parameters. When a code such as CCFM.VENTS is installed on a new machine, the installer must go into these three routines and activate the data statements corresponding to his machine. This involves changing a few comment statements into lines of code.

The user interface of CCFM.VENTS is presently designed to run on all computers that support standard FORTRAN 77. Since screen graphics is not a part of the ANSI FORTRAN 77 standard, it is not possible to implement a "full-screen[1]" graphics interface (input and output) that will run on a wide range of computers. One then has to write a different program for each computer to handle its specialized graphics. The present strategy is to have one version of CCFM.VENTS that would run on any computer that supports FORTRAN-77. Another strategy is to divide CCFM.VENTS into three parts: a Graphics Input Interface, the Numerics Core, and the Graphics Output Interface. The Numerics Core will be the same for all machines except for the slight differences in **I1MACH**, **R1MACH**, and **D1MACH**. The front and back end portions of CCFM.VENTS would then be customized to take advantage of the graphics available on the host computer.

As stated earlier another reason for installing a computer program on several computers is to make the program more robust, i.e. run better. There are subtle differences in how FORTRAN programs run on different computer systems. Even if they all support standard FORTRAN 77! Errors that one computer system will tolerate will be caught by another. One example of this phenomena of programs running differently on computers that support standard FORTRAN is in the area of how computers initialize memory. There are three commonly used methods that computers use to deal with memory that a program is about to use.

- **load memory with indefinites** - The computer's operating system loads the programs "data area" with a value unlikely to be normally used. This value is referred to as an indefinite. The operating system aborts your run if an arithmetic

[1] A full-screen graphics program accesses terminal hardware such as cursor keys, the home key, a mouse etc.

operation is performed using an indefinite value. This method is the default behavior of the CYBER 855.

- **load memory with zeros** - The computer's operating system sets the programs "data space" to zero. This method is the behavior of the Concurrent 3252, IBM PC and most UNIX work stations.

- **keep memory the same** - The computer's operating system does nothing with the programs "data space". This method is the behavior of the Apple Macintosh.

A program designed assuming that memory is pre-loaded with zeros will not run on machines that leave memory alone or load it with indefinites. A program designer would not normally depend on computer characteristics such as zero memory pre-loading. This feature is dangerous to depend on since it could change at any time.

1.2.2 Precision

This report will not debate whether 32 bits (single precision) is sufficient or 60-64 bits (full or double precision) is necessary for scientific computing in general or fire modeling in particular. That is the subject for another report. Rather, the program precision guidelines are presented to allow one to switch easily between single and double precision. If one starts out using single precision but finds that double precision is necessary then the conversion will be easy. This illustrates another principle of good programming practice: "keep your options open." The guidelines to achieve this are given by:

Program Precision Guideline Purpose of guideline: To simplify the conversion of CCFM.VENTS from single to double precision

1. every subroutine should have an IMPLICIT statement at the beginning; IMPLICIT REAL (A-H,O-Z) for the real version of CCFM.VENTS and IMPLICIT DOUBLE PRECISION (A-H,O-Z) for the double precision version.
2. Do not pass floating point constants to subroutines. Instead set a variable equal to the constant and then pass that variable
3. All floating point constants used in the program should be of type double precision, e.g. 1.0D0 not 1.0
4. For floating point variables do not over-ride the default type, i.e. variable names beginning with I, J, ... N should always be of type integer.
5. When using intrinsic functions (SIN, COS) always use the generic form (not DSIN, DCOS, DABS, etc.).

If the above guidelines are followed then to switch precision one simply changes every statement of the form

IMPLICIT REAL (A-H,O-Z)
to
IMPLICIT DOUBLE PRECISION (A-H,O-Z)
or visa versa.

1.2.3 Readability/Maintainability

A set of guidelines have been developed to improve the readability and maintainability of CCFM.VENTS. As a software product matures more and more resources are devoted to its maintenance rather than its design. To minimize the maintenance effort it is important to get the design right in the beginning. Note that maintenance of a program is not simply the job of finding and fixing bugs. It also includes updating and enhancing a programming product. These guidelines are not followed as strictly as the others and are given by:

Readability/Maintainability Guideline: Purpose of guidelines - To improve the readability and maintainability of CCFM.VENTS

1. calling subroutines with many arguments - put input arguments first, followed by output arguments. In the sixth column (continuation column) put an 'I' for lines containing input arguments and an 'O' for lines containing output arguments
2. documenting subroutines - Each subroutine shall contain a prologue describing the use and purpose of the subroutine. It should have the following items in the prologue.
   a. The first card in the prologue should be exactly C*BEG. This signals to the program documentation generator that the prologue is beginning.
   b. a short description of the program
   c. a description of each variable in the calling list and whether it is an input, input/output or output variable
   d. The last card in the prologue should be exactly, C*END. This signals to the program documentation generator that the prologue is finished.
3. Indent DO-loops and IF blocks in order to improve readability
4. When making a program change to improve efficiency weigh carefully its effect on program clarity

1.2.4 Modularity

The modularity guidelines are restricted to the coding of physical subroutines. This guideline is key to the success of CCFM.VENTS for it allows for the import and export of routines by other fire scientists. These guidelines are given by:

Physical Routine Guideline: Purpose of guide-line: To allow other physical scientist to incorporate their work into CCFM.VENTS with a minimum of effort and to allow CCFM.VENTS physical routines to be used in other modeling projects and not through a common block.

- The physical subroutine should not depend on any data structure that exists within CCFM.VENTS. Any quantities needed by the physical routine should be passed to it through the argument list.

1.3 CCFM.VENTS Physical Units

Unless stated otherwise all physical units used in CCFM.VENTS are Scientific International (S.I.). Some of the units used are listed in Table 1.4.1.

Table 1.4.1 CCFM.VENTS Physical Units

quantity	Unit
length	[m]
area	[m^2]
volume	[m^3]
mass	[kg]
density	[kg/m^3]
product concentration	[unit of product/kg of total mass]
pressure	[p] = [N/m^2] = [$kg/(m\ s)^2$] note: 101325 [p] ≈ 1 atmosphere
temperature	[K]
power	[w]
time	second

2. CCFM.VENTS Structure

2.1 Overview of Program Structure

CCFM.VENTS consists of program structures and data structures. Figure 2.1.1 diagrams the major program structures in CCFM.VENTS. The INPUT module consists of those routines that solicit input from the user and transfer data that the user enters to the appropriate data structures. The MODEL module is calculates the desired fire scenario as entered by the user. The OUTPUT module creates reports or summaries of the simulation results in a form that is accessible to one familiar with fire science.

There are three classes of physical routines used in the MODEL module in CCFM.VENTS as illustrated in Figure 2.1.2. These are 1. the physical routines, 2. the routines that compute the ordinary differential equations (ODE) right hand side (rhs) and 3. the physical interface routines. These three routine classes are indicated in the MODEL module portion of Figure 2.1.1.

The ODE rhs routines are **FBUILD** and **FROOM**. **FROOM** calculates the ODE rhs for one room while **FBUILD** calls **FROOM** for each room. The physical routines, **COMWL1**, **VENTHP**, **VENTF**, **FANRES** and **PLUGO** do not depend on any CCFM.VENTS data structures, since all values required by these routines are passed to them through their argument lists. The physical interface routines, **UVENT**, **FVENT** and **FPLUME**, provide the needed variables by accessing the appropriate CCFM.VENTS data structures. New phenomena may be added to CCFM.VENTS by writing subroutine that implements the new phenomena and calling it from **FBUILD**.

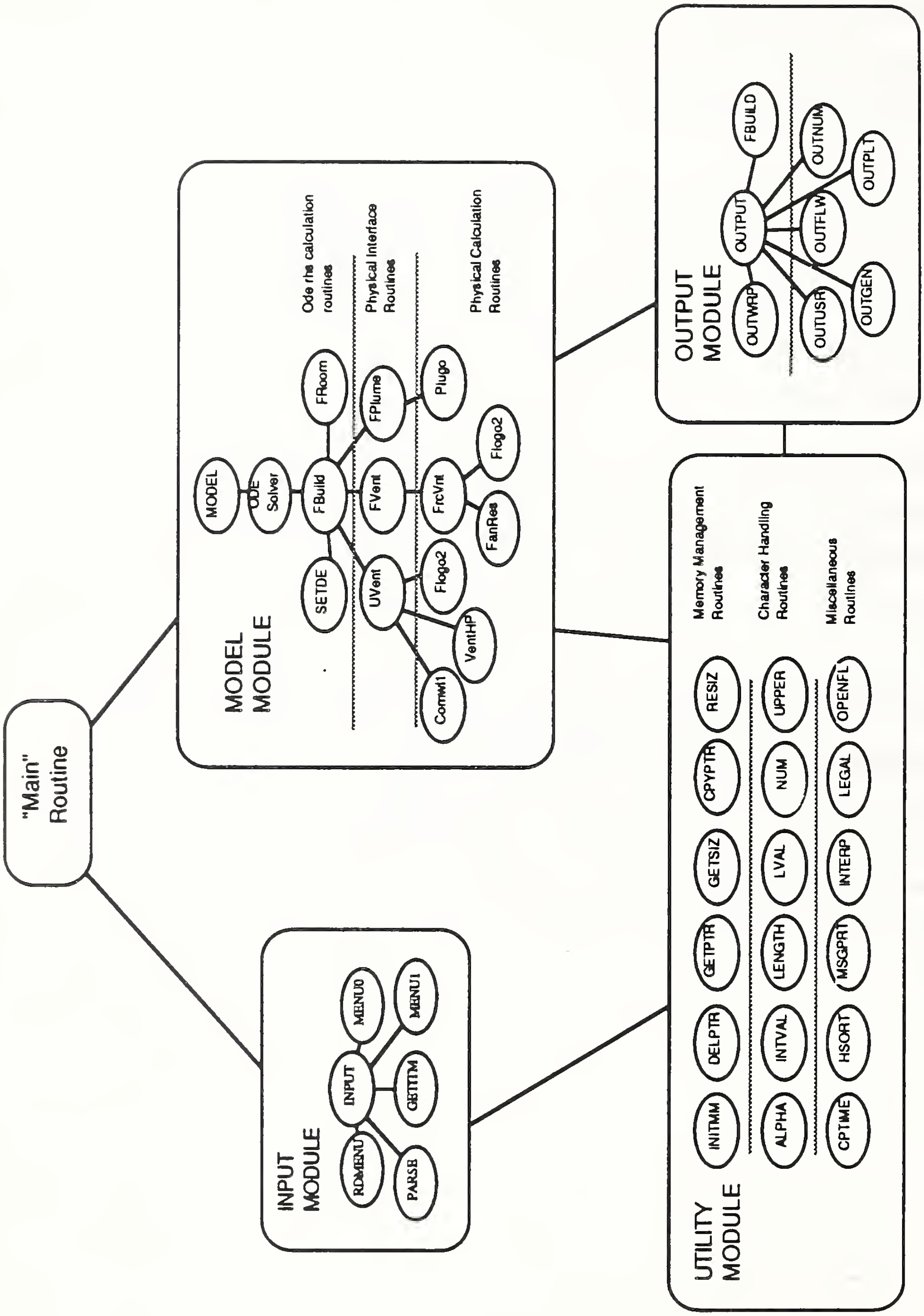


Figure 2.1.1 CCFM.VENTS Structure

CCFM.VENTS has parameters used to define the sizes of various features of the model. These features are the maximum number of rooms both inside and outside and the maximum number of vents. An inside room is distinguished from an outside room by the fact that variables describing the environment in inside rooms are computed and generally changing, while those variables for outside rooms are completely specified. An outside room is assumed to be so large that no matter what happens, i.e. how big the fire gets, the conditions in this room such as pressure or temperature will not change. These quantities are set at compile time and are given in Table 2.1.1. A parameter beginning with MX denotes a maximum size while a parameter beginning with N denotes the actual size of a quantity used in CCFM.VENTS. These quantities are found in the SIZE common block.

Table 2.1.1 CCFM.VENTS Parameter Bounds and Sizes

CCFM.VENTS name		bound	description
MXIRM	(NIRM)	9	maximum (actual) number of inside rooms
MXORM	(NORM)	4	maximum (actual) number of outside rooms
MXTRM	(NTRM)	13 = MXIRM + MXORM	maximum (actual) total rooms = MXIRM + MXORM
MXVNTS	(NVENTS)	20	maximum (actual) number of vents (unforced+forced)
MXFIRE	(NFIRE)	1	maximum (actual) number of fires
MXPRD	(NPROD)	3	maximum (actual) number of products of combustion
MXRMDE	(NRMDE)	10 = 4 + 2*MXPRD	maximum (actual) number of ODE's per room
MXXDES	(NDES)	90 = MXIRM*MXRMDE	maximum (actual) number of ODE's

2.2 Input Module

The input module of CCFM.VENTS diagrammed in Figure 2.1.1 is responsible for soliciting input from the user and placing this input into the appropriate data structures. The input module was designed to run on any computer that supports ANSI-Standard FORTRAN 77. It can also exist independent of CCFM.VENTS as an input module for other programs.

The INPUT module reads in a definition of CCFM.VENTS's menus from the menu description file using the subroutine **RDMENU** . This file is named MENU790 (for July 1990) in the present version of CCFM.VENTS. This definition file gives the names and help information for each CCFM.VENTS command and a set of default data. Commands typed in by the user (or read in from a file) are split apart with **PARSE** and decoded with **GETITM.** The command is interpreted with **MENU1** if the command is specific to CCFM.VENTS, and **MENU0** if the command is not. The only action that is specific to CCFM.VENTS is the work performed by **MENU1** transferring data entered by the user to CCFM.VENTS data structures. This routine is the only one that has to be re-written when using the INPUT module in another program.

The key data structures used by the INPUT module are listed in Table 2.2.1. The INPUT module calculations are detailed in Algorithm 2.2.1. The calculations for each major program structure of CCFM.VENTS are detailed in algorithmic from as in Algorithm 2.2.1. These listings serve as a guide for the person who wishes to examine the corresponding FORTRAN source. The numbers of the form n.m indicate the order in which the steps are performed

Algorithm 2.2.1   Input Module Calculations

```
1.0   Set menu flags, constants
2.0   Open menu definition file, menu help file
3.0   Call RDMENU to read in menu definition file and process the following menu definition commands
      3.1   .NM         new menu
      3.2   .NC         new menu column
      3.3   .C          new command
      3.4   .HB, .HE          indicates the beginning and ending of help information for the previously entered
                              command
      3.5   .PDB, .PDE        indicates the beginning and ending of parameter description for the previously
                              entered command
      3.6   .DF         load default data, i.e. process CCFM.VENTS commands that define the default scenario
4.0   Process a CCFM.VENTS command
      4.1   Read in a card image from the input file
      4.2   Call PARSE to split a  card image into a collection of tokens
      4.3   Call GETITM to identify the command by examining the first token. Handle the token in one of the
            following three ways
            4.3.1   unknown command - inform user that token could not be identified and try again
            4.3.2   generic command - Call  MENU0 for handling generic commands
            4.3.3   CCFM.VENTS command - Call MENU1 for handling CCFM.VENTS commands
      4.4   Return to the calling routine and execute the model if requested by MENU1 otherwise go back to
            4.1 and read another card
```

Table 2.2.1 Key Input Module Data Structures

variable	usage
CARD	Character variable of length 80 containing the current input card image
SB, SE	Integer arrays of length 40 recording the beginning and ending of each token found on the current input card image
NTOK	Number of tokens found on the current input card image
MMENU	I/O Unit number of menu definition file
MHELP	Unit number of menu help file
IBATCH	IBATCH = 1 ==> running in batch mode (non-interactively). End of files are treated differently when running in batch
ILOAD	ILOAD = 1 ==> input data being read in with a LOAD command from a file
IBRIEF	IBRIEF = 1 ==> Menu's are not printed out after each CCFM.VENTS command is executed
IOLEVL	Minimum message level printed by MSGPRT-

2.2.1 Menu Definition File

A principle of good programming practice is to separate the program from the data. A simple case of this idea is to use a **READ** statement (READ(5,*) A) instead of an **ASSIGNMENT** statement (A = 5.) to initialize program data. A program can run more than one case without re-compiling the source code when it uses a READ statement to initialize data.

At the onset of the CCFM.VENTS project it was anticipated that a series of CCFM.VENTS software products would be produced over a period of time. Since the functionality of the INPUT module would remain the same it was decided to use essentially the same INPUT module in each CCFM.VENTS product. The differences would then be stored in a data file known as the Menu Definition File (MDF). The MDF performs the following function:

- defines spelling of a command
- defines arrangement of the menus, i.e. what row and column a command will occur
- specifies help information available to the user through the **HELP** command
- defines parameter descriptions used with the SCREEN mode version of a command

The listing of the MDF used for CCFM.VENTS is given in Appendix C. The help text and parameter description information is identical to information given in the command glossary section of the Users Guide.

The subroutine **RDMENU** reads in this file interpreting the parameters, .NM, .NC, .C, .HB, .HE, .PDB, .PDE and .DF as defined in Algorithm 2.2.1. The order that the .NC (new column) and .C (new command) parameters are placed in the MDF is the same as the arrangement of commands in the printed menu's. The following commands placed in the MDF

```
.NM   TEST MENU
.NC   INPUT
.C    COM1
.C    COM2
.NC   OUTPUT
.C    COMA
.C    COMB
```

will result in the following menu:

```
     TEST MENU
INPUTOUTPUT
COM1        COMA
COM2        COMB        .
```

The .DF command is used to load default data into CCFM. By changing this portion of the MDF, you can change the default fire scenario from a one-room, 1 vent, 250,000 watt fire to a five room, 10 vent, 1,000,000 watt fire. To change CCFM.VENTS's default fire scenario, use CCFM.VENTS to input the scenario you want. Then use the DUMP command described in the User's Guide to create an ASCII text file of CCFM.VENTS commands for this case. Use a text editor to place this scenario in the MDF in place of the former one.

All help information about a command is stored in a temporary data file which is discarded when CCFM.VENTS begins the modeling step. An abbreviated listing of the MDF, MENU989 is given in Appendix F.

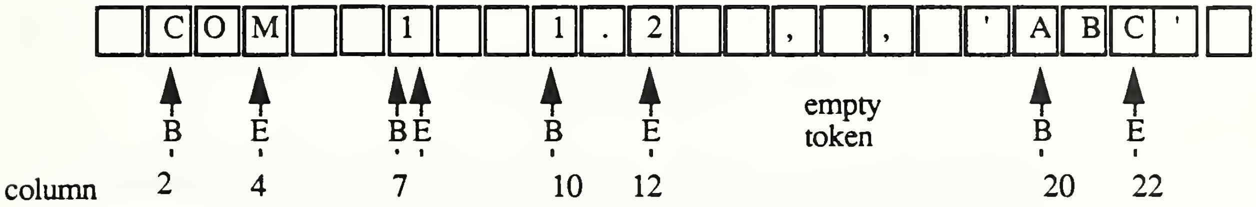
2.2.2 Parsing Input

The workhorse of the INPUT module is a set of routines that parse or split a character string into a series of tokens. Each token is a contiguous set of non-blank characters or characters enclosed by quotes as illustrated by Figure 2.2.2.1. By performing this operation the user is not required to enter information in certain specified columns or fields. The input requirements of CCFM.VENTS are completely free-format.

Consider the following character string named CARD.

COM 1 1.2 , , 'ABC'.

As Figure 2.2.1.1 illustrates, the above command line has 5 fields. The fourth field is empty. This could correspond to an option that the user did not select or did not wish to change.



Token	1	2	3	4	5
CCFM data structure	SB(1) = 2 SE(1) = 4	SB(2) = 7 SE(2) = 7	SB(3) = 10 SE(3) = 12	SB(4) = 0 SE(4) = undefined	SB(5) = 20 SE(5) = 22

SB(i) = column number of start of i'th token
SE(i) = column number of end of i'th token

Figure 2.2.2.1 Parsing a CCFM.VENTS command

The subroutine **PARSE** is passed a character string named CARD. This string could have been entered by the user or constructed in some other part of CCFM.VENTS. **PARSE** returns values in two arrays SB and SE which denote the beginning and ending of each sub-string in CARD. In addition the number of tokens or sub-strings is found. This information is used by other portions of the INPUT module to

- interpret CCFM.VENTS commands
- convert character data to floating point, integer or logical data types

2.2.3 Handling a CCFM.VENTS Command

There are three ways that a CCFM.VENTS command may be initiated

- a command may be typed in at a computer terminal
- a command may be read in from a data file. CCFM.VENTS executes commands given in a file that were specified with a LOAD command.
- A command may be constructed from input that a user gave while in screen mode.

The last two cases for handling CCFM.VENTS commands are really a special case of the first. The LOAD command is implemented simply by re-directing the input from the terminal screen to the file specified by the user, i.e. the I/O unit number for INPUT's read statement is changed to point to the LOAD file instead of the terminal screen. Screen mode is implemented by constructing a command-line equivalent to the screen inputs the user gave.

The operations performed while handling a CCFM.VENTS command are given in step 4.0 of Algorithm 2.2.1. Parsing a CCFM.VENTS command was discussed in section 2.1.

The subroutine **GETITM** attempts to interpret the first token of a command line as a CCFM.VENTS menu command. This routine has an alphabetical list of legal CCFM.VENTS commands as specified with the menu definition file. **GETITM** uses a binary search to try and match the first token with a valid CCFM.VENTS command. If **GETITM** is unable to find a match then it prints a message asking the user to enter the command again.

All of the work performed so far in the INPUT module is independent of CCFM.VENTS. Once a command has been identified the routine **MENU1** is called to implement that command. This routine uses utility routines described in section 2.5 to transfer data from the card image entered by the user to data in internal CCFM.VENTS data structures.

2.3 Output Module

The output module diagrammed in Figure 2.1.1 is responsible for displaying results to the user. This display should be readable to one familiar with fire science though not necessarily familiar with the inner workings of CCFM.VENTS.

CCFM.VENTS calculates more values than can be displayed effectively in one report. Further, some numbers that are calculated may not be of interest to every one. The obvious solution is to have several reports that can be selected by the user at the beginning of the run. The user then selects only those reports of interest.

The routine **OUTPUT** is called at the end of each solver time step. **OUTPUT** calls **FBUILD** to initialize global data for the current solution vector. It then calls the report routines: **OUTUSR**, **OUTNUM**, **OUTGEN**, **OUTPLT**, **OUTFLW** depending on what report options were selected by the user. These calculations and a clarification of the terminology used to name the above subroutines are detailed in Algorithm 2.3.1. The key data structures used in the output module are given in Table 2.3.1.

To investigate the numerical properties a routine, **EIGF**, was written to calculate the Jacobian and its eigenvalues of the rhs of the ODE. The eigenvalues measure the stiffness and stability of the particular fire scenario being solved. Eigenvalues with positive real parts indicate instability. If all their real parts are negative and their magnitudes range over several orders of magnitude then the ODE problem is usually thought of as stiff. The procedure for calculating the Jacobian's eigenvalues is the same for any differential equation. First, the Jacobian is approximated using difference quotients by calling the routine (FBUILD in the case of CCFM.VENTS) that computes the right hand side of the differential equation NDES + 1 times where NDES is the number of differential equations. Then standard software, for example a routine from Eispack, is called to compute the Jacobian's eigenvalues

Table 2.3.1 Key Output Module Data Structures

variable	usage
RPTS	Integer array of size MXRPT=6. If RPTS(I) = 1 then output for report I was requested. Reports 1 through 6 are 1 - **OUTGEN**, 2 - **OUTPRD**, 3 - **OUTFLW**, 4 - **OUTPLT**, 5 - **OUTNUM**, 6 - **OUTUSR**
INSTEP	Number of solver steps since the last printed output
IPSTEP	Number of printed steps so far
CPP	Cumulative CPU time in seconds since the beginning of the run
CPST	CPU time for this step
PRTIM	Flag set to 1 if **OUTPUT** should print reports

While the Model is running, all output is printed to a temporary report file. It will typically look like

```
A    An Entry for Report 1
B    An Entry for Report 2
C    An Entry for Report 3
A    Another Entry for Report 1
B    Another Entry for Report 2
C    Another Entry for Report 3
```

Notice that the above listing is displayed in the same order that it was produced. It is not in the order that you would want to read it. So a special code is placed in column 1; A for report 1, B for report 2, C for report 3, etc. At the end of the model run these codes are used to un-scramble the temporary report file to produce an output listing that is in order as in

```
     An Entry for Report 1
     Another Entry for Report 1
     An Entry for Report 2
     Another Entry for Report 2
     An Entry for Report 3
     Another Entry for Report 3
```

This scheme is easily extended. If the user of CCFM.VENTS wishes to add a report for quantities calculated by CCFM.VENTS but not presently printed she/he simply calls her/his new report routine within **OUTPUT**, e.g. after entry 3.2.5 in Algorithm 2.3.1. This new output routine must output a unique code in column 1. The column 1 codes presently used in CCFM.VENTS are : A - OUTGEN, B - OUTPRD, C - OUTFLW, D - OUTNUM, E - OUTUSR. Note that the subroutine **OUTUSR** is not presently used and could be used to implement a user defined output report.

Algorithm 2.3.1 Output Module Calculations

```
1.0    Update timing variables, step numbers
2.0    Call FBUILD to update ODE information
3.0    If appropriate flags are set:
       3.1    Call EIGF to calculate ODE Jacobian and its eigenvalues
       3.2    Produce entries in temporary output file. Each of the following subroutines precedes the text to
              appear in the report with a code. The code is the character A for OUTGEN, B for OUTPRD, C for
              OUTFLW, D for OUTNUM and E for OUTUSR. These codes are used by OUTWRP (output
              wrapup) to unscramble the temporary report file.
              3.2.1    Call OUTGEN to produce general report entry
              3.2.2    Call OUTPRD to produce product of combustion report entry
              3.2.3    Call OUTFLW to produce flow report entry
              3.2.4    Call OUTNUM to produce numeric report entry
              3.2.5    Call OUTUSR to produce user defined report entry
4.0    Call OUTWRP at the end of the simulation to sort the temporary output file by report type
```

CCFM.VENTS will automatically split out this new report along with all the others at the end of the model run as long as the column 1 code is chosen uniquely (not A, B, ...,or E). This splitting is done at the end of the simulation by the subroutine **OUTWRP** (output wrapup).

2.4 Model Module - The Governing Equations and Their Solution

The model module implements the physical equations described in volumes I and III of this report. These equations are required to compute the right hand side (rhs) of the ordinary differential equation (ODE)

$$dU/dt \equiv U' = f(t,U) \qquad (2.4.1)$$

$$U(t_0) = U_0$$

where t is the independent variable, time; U and U' are both real-valued vectors of length NIRM*(4+2*NPROD); where NIRM is the number of inside rooms in the simulation; and NPROD is the number of products of combustion including oxygen. The format for both U and U' for the first room is given in Figure 2.4.1.2. The ODE of Eq. (2.4.1) corresponds to Eqs. (2.3.2), (2.3.3), (2.3.4') and (2.3.6)-(2.3.8) of Part I [10]. The solution variables stored in U are:

relative pressure at the floor, layer interface elevation, upper/lower layer total mass and lower/upper layer total product mass for each room. The ODE of Eq. (2.4.1) is discussed in section 2.4.2 and derived in Part I of this report. The data structures used by the Model Module are given in Table 2.4.1.

Table 2.4.1 Key Model Module Data Structures

PRINTF	Contains name of temporary output file
PRINTG	Contains name of final output file
PLOTF	Contains name of plot file
DUMPF	Contains name of dump file
Y	Contains ODE solution
TSTART	Initial time of simulation
TCUR	Current time of simulation
TSTOP	Final time of simulation

The routine **FBUILD** calculates the rhs, f(t,U), of Eq. (2.4.1). The ODE solver then is responsible for stepping the solution of Eq. (2.4.1) forward in time.

The pressure in each room adjusts rapidly to changes in the room environment compared to other quantities such as layer height or temperature. It is this property of Eq. (2.4.1) that makes its solution difficult. For some, but not all, fire scenarios the CCFM.VENTS ODE's are stiff. Stiff ODE's require special solution methods. Standard explicit ODE algorithms such as Runge-Kutta or Predictor-Corrector methods such as Adams-Bashforth are well known to be unsuitable for the solution of stiff ODE's. This is because the transient component of the ODE, in this case the pressure, causes these types of ODE methods to have small regions of stability. As a result these methods required small step sizes. Stability of an algorithm refers to how error propagates; the error that occurs at each step. A stable algorithm damps error while an unstable algorithm allows it to grow.

Stiffness is well explained in the survey articles by Shampine [3] and Byme [4]. It is also discussed in terms of a fire model in section 2.4.4. The CCFM.VENTS ODE equation solver uses a backward difference formula first proposed by Gear [5] and implemented by Gear [6], Watts and Shampine [7], and others. These algorithms have a large stability region for stiff problems. The trade off is that a system of linear equations needs to be solved at each step. While this requires more work per time step than standard methods for non-stiff problems, use of these algorithms can pay off as a result of the larger step sizes that can be taken. The calculations performed by the Model module are given in Algorithm 2.4.1.

Algorithm 2.4.1 Model Module Calculations

```
1.0    Open files for INPUT, OUTPUT and plot data
2.0    Initialize solver flags, timing variables, error tolerances
3.0    Save ODE solution from last time step
4.0    get elapsed CPU time from the beginning of the run
5.0    Call the ODE solver, DEBDF, to get the solution at next time step
6.0    if there are solver errors then handle them and try again
7.0    Call OUTPUT to produce entries in the output file
8.0    Call DUMP to produce an entry in the dump file
```

2.4.1 FBUILD

The routine **FBUILD** diagrammed in Figure 2.1.1 is called by CCFM.VENTS's ODE solver in order to compute the ODE rhs given by Eq. (2.4.1). The routine **FBUILD** then calls the physical interface routines, **FVENT**, **FPLUME**, **UVENT**, in order to compute mass/enthalpy/product of combustion flows. These flows are then used by **FROOM** to compute the ODE rhs. The ODE rhs is returned to the solver so that it can estimate the solution at the next time step.

Densities were used as solution variables in the original ODE formulation. Moss [8] proposed that mass would be better numerically since the mass equation did not have a re-movable singularity (layer volume going to zero) in the denominator. An interface between the ODE data structure illustrated by Figure 2.4.1.1 and the variables used by the physical interface routines was instituted in order to reduce the impact of any future change of the ODE formulation . The physical interface routine's do not access the ODE variables directly. The routine **SETDE** implements this interface by copying the information stored in the ODE data structure into variables used by the physical interface routine. If the ODE formulation changes in the future, only the routine **SETDE** has to be changed.

The data in the solver solution array, y, is organized by room. Each room requires 4 + 2*NPROD locations in the solver array. The definition of each memory location is identified by Figure 2.4.1.1.

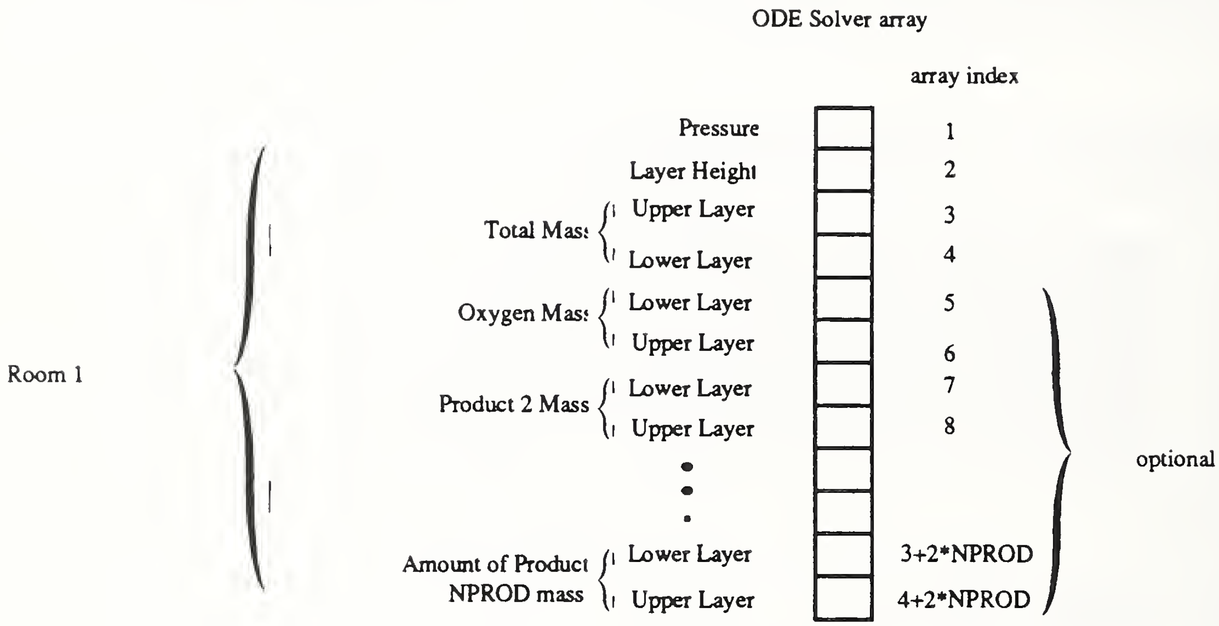


Figure 2.4.1.2 ODE Solver Array Structure For One Room

Each physical interface routine routine, (FVENT, UVENT and FPLUME) calculates flow due to: mass, enthalpy and product of combustion. Rather than using six variables ((2 layers) x (3 flow types)) it was decided to organize the flow results into one 3-dimensional array data structure. This structure is given in Figure 2.4.1.3. The arrays FLWP, FLWF, FLWU and FLWTOT are declared in the common block /FLOWS/. Each of these arrays has the structure as defined in Figure 2.4.1.3. Further, FORTRAN parameter statements are used define the integer symbols, L, U, M, Q, P to: L=1, U=2, M=1, Q=2 and P=3 to increase the program's readibility. So FLWF(3,Q,L) refers to the forced enthalpy flow to the lower layer of the third room.

The calculations performed by **FBUILD** are listed in Algorithm 2.4.1.3.

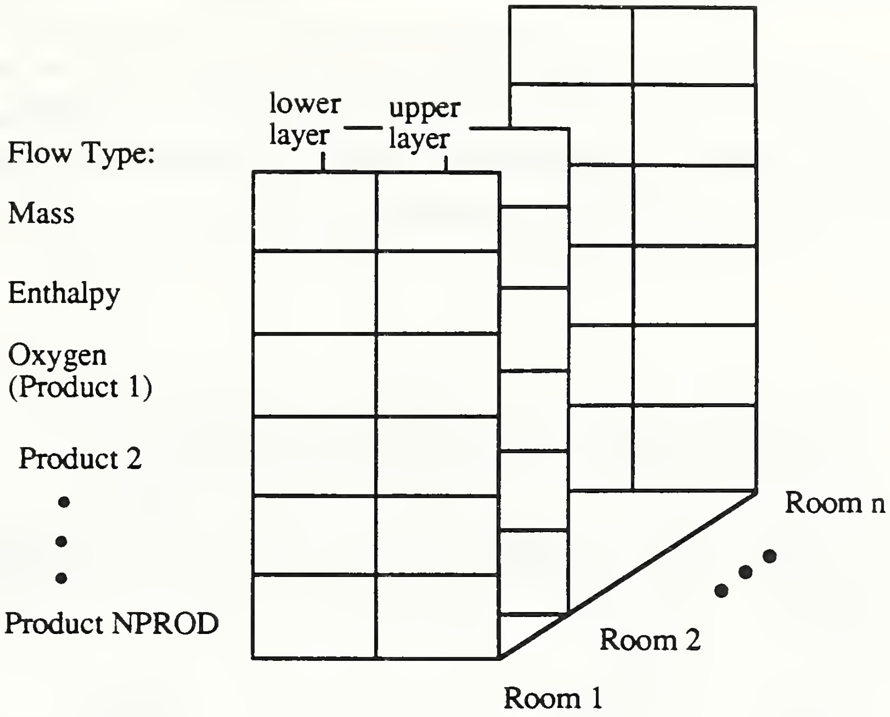


Figure 2.4.1.3 Data Structure for Storing Flows

Algorithm 2.4.1.1 FBUILD Calculations

```
1.0    Call SETDE to convert from ODE to CCFM.VENTS.VENT style data structures
2.0    Calculate mass, enthalpy, product of combustion flows
       2.1    Call UVENT to compute flows for natural vents, return flow in FLWU
       2.2    Call FVENT to compute flows for forced vents, return flow in FLWF
       2.3    Call FPLUME to compute flows for fire plumes, return flow in FLWP
3.0    Sum flows from sub-models listed in 2.1-2.3, sum corresponding components of FLWU, FLWF, FLWP and
       place total into FLWTOT
4.0    Call FROOM to calculate derivatives of pressure, layer height, upper/lower layer mass and upper/lower
       layer product equations for each room
```

2.4.2 FROOM - Calculating the ODE rhs for an Arbitrary Room of the the Facility

Figure 2.4.2.1 illustrates the basic features simulated by CCFM.VENTS for one one room. A generic fire environment in a room of a modeled facility consists of a fire and its associated plume, natural and forced vents, and an upper and lower layer of elevated temperature, product-of-combustion contaminated layers separated by a layer interface.

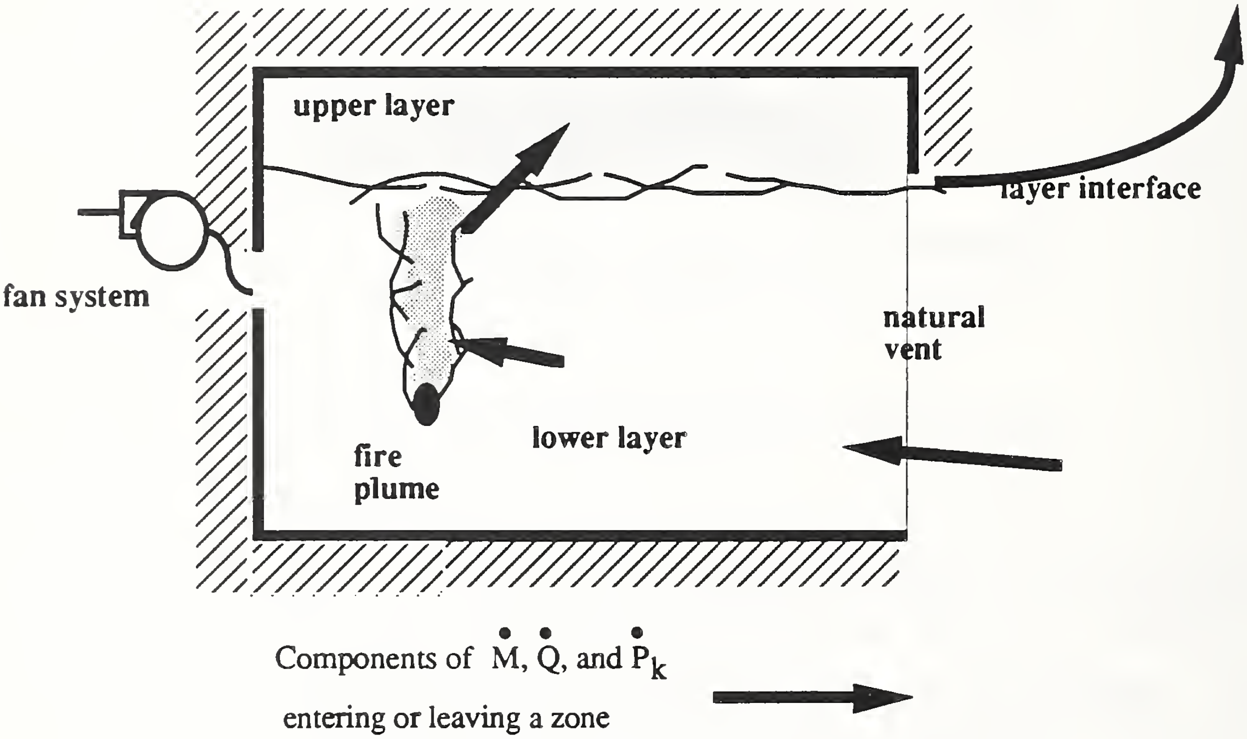

upper layer
layer interface
fan system
natural vent
lower layer
fire plume
Components of $\dot{M}$, $\dot{Q}$, and $\dot{P}_k$ entering or leaving a zone

Figure 2.4.2.1 Zone Model Features

The flow through a natural vent is computed using **UVENT**. The flow is computed by using the cross-vent hydrostatic pressure-difference profile and implementing Bernoulli's equation. The pressure-difference profile is computed from the densities of the two layers and the pressures at the floors of the two rooms, on either side of the vent. The flows due to a forced vents is computed using **FVENT**, and flow due to a fire plume is computed using **FPLUME**.

The subroutine **FROOM** is responsible for calculating the rhs of the (4 + 2*NPROD) components of the ODE of Eq. (2.4.1) associated with (4 + 2*NPROD) solution variables which describe the two-layer fire environment in an arbitrary room of the modeled facility. For a given time step in the calculation, **FROOM** is called for each inside room of the modeled facility from room 1 to room NIRM.

The solution variables for arbitrary room i are: $\delta p_{FLOOR,i}$ pressure at the floor above the datum pressure, p_{DATUM}; $y_{LAYER,i}$, elevation of the layer interface; $M_{U,i}$ and $M_{L,i}$,the total mass in the upper and lower layer, respectively; and $P_{k,U,i}$ and $P_{k,L,i}$, k = 1 to NPROD, the total amount of product k in the upper and lower layer, respectively. The ODE's for the solution variables of room i are reproduced here from Section 2.3 of Part I [10].

perturbation pressure at the floor:

$$\frac{d\delta p_{FLOOR,i}}{dt} = \frac{\gamma - 1}{V_i}\left(\dot{Q}_{U,i} + \dot{Q}_{L,i}\right)$$

layer interface elevation:

$$\frac{dy_{LAYER,i}}{dt} = \frac{(\gamma - 1)\left((y_{CEILING,i} - y_{LAYER,i})\dot{Q}_{L,i} - (y_{LAYER,i} - y_{FLOOR,i})\,\dot{Q}_{U,i}\right)}{\gamma V_i\left(p_{DATUM} + \delta p_{FLOOR,i}\right)}$$

mass in upper and lower layer:

$$\frac{dM_{U,i}}{dt} = \dot{M}_{U,i}$$

$$\frac{dM_{L,i}}{dt} = \dot{M}_{L,i}$$

amount of product k in upper and lower layer:

$$\frac{dP_{k,U,i}}{dt} = \dot{P}_{k,U,i}$$

$$\frac{dP_{k,L,i}}{dt} = \dot{P}_{k,L,i}$$

In the above, $\dot{M}_{U,i}$ [$\dot{M}_{L,i}$] and $\dot{P}_{k,U,i}$[$\dot{P}_{k,L,i}$] are the net rates of mass and product-of-combustion k flowing to the upper [lower] layer of room i, respectively. Also, $\dot{Q}_{U,i}$ [$\dot{Q}_{L,i}$] is the net rate of enthalpy plus heat transfer plus energy release flowing to the upper [lower] layer of room i. Finally, V_i is the volume of the room.

The above equations are general in that the above net flow rates represent the sum of transfers to the layers from all plumes, jets, near-boundary flows, combustion zones, and other isolated or distributed sources considered and taken into account in any particular CCFM.VENTS application. The reader is referred to Part I [10] of this work for the details of the physical basis of these transfers for CCFM.VENTS.

The rhs of the above equations are computed and stored in the array UPRIME. The elements of this array corresponds to the components of U' of Eq. (2.4.1) for an arbitrary room. Thus, UPRIME is of length NRMDE; the first four positions of UPRIME are $d\delta p_{FLOOR,i}/dt$, $dy_{LAYER,i}/dt$, $dM_{U,i}/dt$, $dM_{L,i}/dt$; and each subsequent position pair, up to the NPROD pair of products being simulated, is $dP_{k,U,i}/dt$ and $dP_{k,L,i}/dt$.

The structure of U of Eq. (2.4.1) corresponds exactly to the structure of UPRIME. If an entry of U contains a given variable, then the corresponding entry in UPRIME will contain the derivative of that variable with respect to time. U is a real array of length NEQN = NRMDE. Corresponding to the above equation set, the structure of U is given by:

- $U(1) = \delta p_{FLOOR,i}$, perturbation pressure

- $U(2) = y_{LAYER,i}$, layer height

- $U(3) = M_{U,i}$, upper-layer mass

- $U(4) = M_{L,i}$, lower-layer mass

- $U(3+2*k+1) = P_{k,U,i}$, amount of upper-layer product k

- $U(3+2*k+2) = P_{k,L,i}$, amount of lower-layer product k

The calculations performed by **FROOM** are detailed in Algorithm 2.4.2.1.

Algorithm 2.4.2.1 FROOM Calculations

```
1.0   Convert from CCFM.VENTS style to local FROOM data structures
2.0   Calculate rhs of:
      2.1   Pressure equation
      2.2   Layer height equation
            2.2.1   if layer is below floor and is descending reset derivative to zero
            2.2.2   if layer is above ceiling and is rising reset derivative to zero
      2.3   Upper layer mass - if layer is at or above ceiling and layer is not filling then reset this derivative to
            zero
      2.4   Lower layer mass - if layer is at or below floor and layer is not filling then reset this derivative to zero
      2.3   Upper layer product of combustion - if layer is at or above ceiling and layer is not filling then reset
            this derivative to zero
      2.4   Lower layer product of combustion - if layer is at or below floor and layer is not filling then reset this
            derivative to zero
```

This structure of U is illustrated in Figure 2.4.1.2.

2.4.3 Physical Interface Routines

The Physical Interface Routines presently consist of the subroutines **FVENT**, **UVENT** and **FPLUME**. These routines are the drivers that implement the physics described in volume I by calling the appropriate sub-modeling algorithms described in Part III [11]. For example; **UVENT** calls **COMWL1, VENTHP, FLOGO2; FVENT** calls **VENTF** and **FLOGO2** ; and **FPLUME** calls **PLUGO**. The physical interface routine's are distinguished from the sub-modeling algorithms by the data that they can access. The physical interface routine's access CCFM.VENTS's global data; passing required values obtained from CCFM.VENTS's common blocks to the sub-modeling algorithms. All physical interface routine's return flow information in a data structure that is discussed in Section 2.4.1 and illustrated in Figure 2.4.1.3.

To extend CCFM.VENTS to include other Physical phenomena one simply adds another physical interface routine to **FBUILD**. There must be one physical interface routine for each phenomena modeled. The new routine would be called right after entry 2.3 in Algorithm 2.4.1.1.

2.4.3.1 FPLUME

FPLUME calculates flows caused by fire plumes. Presently the fire plume model is based on the point-source plume model described in section 3.2 of Part I [10] and in the **PLUGO** entry of Part III [11]. The plume model can be modified by replacing the call to **PLUGO** with some other plume algorithm. The calculations performed by **FPLUME** are described in Algorithm 2.4.3.1.1.

CCFM.VENTS presently stores a table of fire sizes and associated times using memory management routines. Memory management routines are discussed in section 2.5.2. **FPLUME** uses **INTERP** to interpolate this table to obtain the fire size (energy release rate) at arbitrary times between table entries. **FPLUME** then passes the fire size along with other needed data, such as the height of the fire above the floor and room of fire origin to **PLUGO**. **PLUGO** in turn calculates mass, enthalpy and product of combustion flows due to the fire to the layers of the room of fire origin.

Algorithm 2.4.3.1.1 FPLUME Calculations

```
1.0   Initialize outputs to zero
2.0   For each fire (presently 1):
      2.1   Call INTERP to calculate the energy release rate of the fire at the given time
      2.2   Record properties such as layer density, heights, fire heights
      2.3   Call PLUGO to calculate mass, enthalpy, product of combustion flow due to the fire plume
      2.4   Accumulate results in the data structures FLWP to be returned to calling routine (FBUILD)
```

2.4.3.2 FVENT

FVENT calculates flow through forced vents. The details of the **FVENT** calculations are presented in section 3.4 of Part I [10] and in the **VENTF**, **FLOGO2** and **FANRES** entries of Part III [11]. CCFM.VENTS models forced ventilation systems made up of a fan, with an associated fan curve, and a duct connecting two rooms. One of these rooms may be an outside environment. A fan curve defines the relationship between the volume flow rate (and direction) through the fan and the pressure rise across the fan. CCFM.VENTS's memory management routines are used to store the fan curve. The number of forced vents plus the number of natural vents must not exceed the parameter MXVNTS which is defined in Table 2.1.1. This is a parameter that can be set at compile time. The calculations performed by **FVENT** are detailed in Algorithm 2.4.3.2.1.

Algorithm 2.4.3.2.1 FVENT Calculations

```
1.0   Convert from CCFM.VENTS to VENTF style data structures
2.0   Call VENTF to calculate mass, enthalpy and product of combustion flows
3.0   Call FLOGO2 to debit flows from "from" room
4.0   Call FLOGO2 to deposit flows into "to" room. Note: two different calls to FLOGO2 are necessary since inlet
      and outlet fan duct system elevations may be different in general.
5.0   Accumulate results for each forced vent into data structures that will be passed back to the calling routine
      FBUILD
```

2.4.3.3 UVENT

UVENT calculates flow in a natural vents. As mentioned earlier, the flow is calculated using Bernoulli's equation and is based upon the cross-vent pressure profile that exists between the two rooms that are connected. The details of the **UVENT** calculation are presented in section 3.3 of Part I [10] and in the **COMWL1, VENTHP,** and **FLOGO2** entries of Part III [11]. As in the forced vent case the number of forced vents plus the number of natural vents must not exceed the parameter MXVNTS which is defined in Table 2.1.1. This is a parameter that can be set at compile time. The calculations performed by **UVENT** are detailed in Algorithm 2.4.3.3.1.

Algorithm 2.4.3.3.1 UVENT Calculations

```
1.0    Convert data structures from CCFM.VENTS to COMWL1/VENTHP/FLOGO2 style
2.0    For each pair of rooms (i,j) that have a natural vent connection:
       2.1    Call COMWL1 to compute set up information required by VENTHP
       2.2    For each vent connection in room (i,j):
              2.2.1    Call VENTHP to calculate slab flows
              2.2.2    Call FLOGO2 to determine where the flows should be deposited (which room, which
                       layer)
```

2.4.4 Solving The ODEs In CCFM.VENTS

Stiff ordinary differential equations (ODE) are an important class of problems that occur when several phenomena being modeled have characteristic time-scales that vary by several orders of magnitude. The system of ODE's used in zone fire modeling are stiff because the pressure adjusts to changing conditions in a fire much faster than other quantities being modeled such as upper, lower layer temperatures or layer heights.

The curious aspect of stiff ODE's is that the solution appears to be in equilibrium yet the computational costs for computing this solution is enormous when using standard algorithms such as Runge-Kutta or Predictor-Corrector algorithms such as Adams-Bashforth. The question then is why does it cost so much to solve a problem whose answers change slowly.

To analyze the numerical character of the ODE's in CCFM.VENTS we have used DEPAC which is a set of three ODE solvers DERKF, DEABM, and DEBDF designed by Shampine and Watts.

DERKF is a fifth order variable step size Runge-Kutta code. It can be used for nonstiff and mildly stiff ODE's when derivative evaluations are not expensive. It should not be used for high accuracy, nor for answers at a great many points. DEABM is a variable order, variable step size Adams code. It can be used for nonstiff and mildly stiff ODE's when high accuracy is required. DEBDF is a variable order, variable step size backward differentiation formula code which can be used on stiff ODE's when moderate accuracy is required. DERKF and DEABM attempt to determine when their use is not suitable by performing diagnostics for stiffness.

When used in CCFM.VENTS both DERKF and DEABM reported that the problem might be stiff. This occurred for a wide range of fire scenarios. In order to get a more quantitative indication of stiffness, we examined the Jacobian of the right hand side of the system of ODE's. A subroutine, **EIGF**, was written to approximate this Jacobian and its eigenvalues. The characteristic time scales of the solution variables and the solution behavior was determined from the eigenvalues. It was found that the characteristic time scale for the pressure variable was much smaller than other solution variables. This difference was found for some problems to be over five orders of magnitude. As a result of this analysis, the stiff ODE solver, **DEBDF** , was chosen.

2.4.4.1 Theoretical Background

Consider the following initial value problem:

$$y'(t) = f(y,t) \qquad (2.4.2)$$
$$y(t_0) = y_0$$

where y(t), y'(t) are real valued N-vectors and f is a real-valued N-vector function. A numerical algorithm for solving Eq. (2.4.2) generates estimates, y_i, to the solution at t_i, $y(t_i)$. The **global error** at t_i is defined by $e_G(t_i) := y(t_i) - y_i$. As Figure 2.4.4.1.1 illustrates, the global error consists of two terms, **local error** and **propagation error**. The local and propagation errors at t_i are defined in terms of an ODE related to Eq. (2.4.2). This "local" ODE has the same right hand side as Eq. (2.4.2) but has different initial conditions as in

$$z'(t) = f(z,t)$$
$$z(t_{i-1}) = y_{i-1}.$$

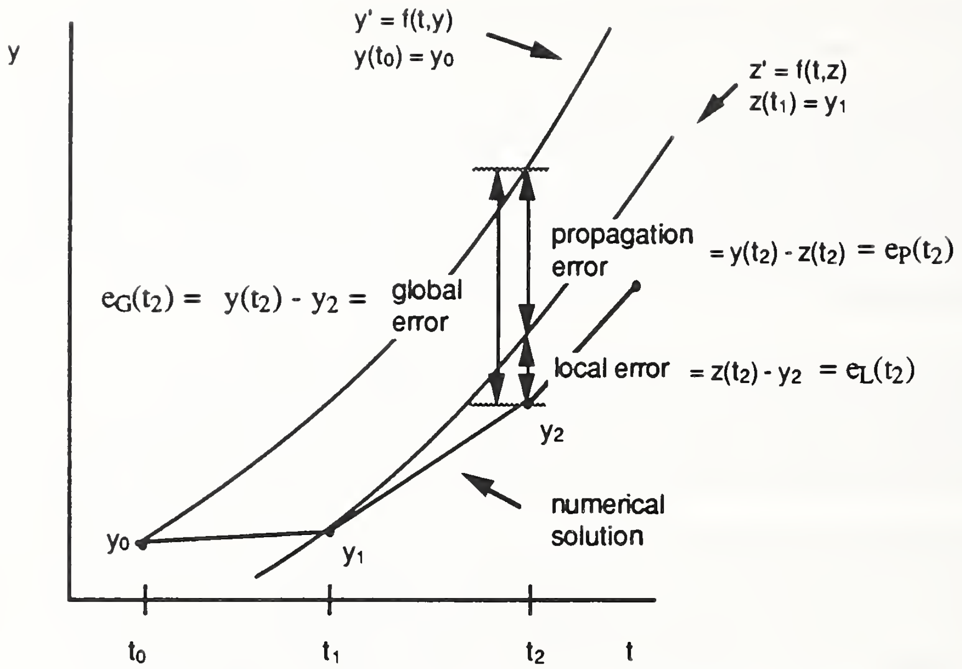


Figure 2.4.4.1.1 ODE Error Terms

The local error is defined by $e_L(t_i) := z(t_i) - y_i$ and the propagation error is defined by $e_P(t_i) := y(t_i) - z(t_i)$. An ODE algorithm whose propagation error term grows at each step is said to be **unstable**. ODE algorithms can be divided into two classes, explicit and implicit. Explicit algorithms, such as DERKF, are not suitable for solving stiff ODE's because of the prohibitively small step size required to maintain stability (keep the propagation error terms small). What distinguishes stiff ODE's from other "hard" ODE's is that for explicit methods the step size required to maintain stability is much smaller than that needed to maintain accuracy. The routines in DEPAC estimate the global error at each step. Local error and propagation errors are not separately estimated, so there is no direct check for instability. Global error is estimated by comparing two estimates having different orders of accuracy. If the global error estimate is too large, step size is decreased. On the other hand, if the global error estimate is too small,step size is increased. Routines DEABM and DEBDF also have the option of changing the order of the method. Roughly speaking, smooth solutions are more efficiently approximated by higher order methods.

Implicit algorithms are used to solve stiff ODE's. After n steps of such an algorithm, approximation y_n at time $t = t_n$ is obtained. The step size Δt for step n+1 is chosen based on the global error estimate from the step n. We illustrate step n for the backward Euler method. If Eq. (2.4.2) is discretized using a divided difference approximation to y' and a trapezoidal rule approximation for f, we arrive at

$$\frac{y_{n+1} - y_n}{\Delta t} = \frac{f(y_n, t_n) + f(y_{n+1}, t_{n+1})}{2} \tag{2.4.3}$$

where y_{n+1} is the unknown approximation to $y(t_{n+1})$ and $t_{n+1} = t_n + \Delta t$. **Functional iteration** or **successive substitution** and **Newton's method** are the two standard techniques for solving the nonlinear equation Eq. (2.4.3) for y_{n+1}. These methods are iterative and their rates of convergence depend on Δt. The usual approach is to iterate a few times and, if convergence has not occurred, decrease Δt and start over. For stiff problems, Newton's method is used because functional iteration will not converge unless Δt is extremely small as we show next.

To apply functional iteration, Eq. (2.4.3) is re-cast into the fixed point equation[2]:

$$y_{n+1}^{(k+1)} = T(y_{n+1}^{(k)}) = y_n + \frac{\Delta t}{2}\left(f(y_n, t_n) + f(y_{n+1}^{(k)}, t_{n+1})\right) . \tag{2.4.4}$$

The k'th iterate, $y_{n+1}^{(k)}$, is updated using the iteration function T to obtain a new iterate, $y_{n+1}^{(k+1)}$. The iteration process is complete if two successive guesses, $y_{n+1}^{(k)}$, $y_{n+1}^{(k+1)}$ are sufficiently close. Now consider the simple model problem, $y' = f(y) = Ay$, $y(t_0) = y_0$ where A is an N by N matrix. This is a linear ODE with solution $e^{A(t - t_0)}y_0$. The iteration function, T, and its Jacobian T' for this model problem are given by:

$$T\left(y_{n+1}^{(k)}\right) = y_n + A\frac{\Delta t}{2}\left(y_n + y_{n+1}^{(k)}\right)$$

$$T'\left(y_{n+1}^{(k)}\right) = A\frac{\Delta t}{2} .$$

[2] Eq. (2.4.4) is called a fixed point equation because a solution, y*, of Eq. (2.4.3) satisfies y* = T(y*). This y* is called a fixed point.

The Jacobian, T', is an N by N matrix whose i-k'th element is given by $\partial T_i/\partial y_k$ where y_k is the k'th component of the vector y and T_i is the i'th component of the vector valued function T. In general, functional iteration will converge linearly to the desired solution if the initial guess, $y^{(0)}_{n+1}$, is sufficiently close and if the Jacobian, T', exists, is continuous, and has a 2-norm less than 1. This result is based on the fixed point theorems presented in Chapter 5 of [9]. For the above model problem, the 2-norm of T' satisfies

$$\| T' \|_2 = \| \Delta t \, A/2 \|_2 = | \Delta t \, \lambda_{max} | /2 < 1$$

only when

$$\Delta t \;<\; 2/ | \lambda_{max} |$$

where λ_{max} is the eigenvalue of A with largest magnitude. Functional iteration will have convergence problems when the eigenvalues of A are large which is the case for stiff ODE's. Stiffness occurs when the eigenvalues of f', the Jacobian of the right hand side of the system of ODE's, have widely varying magnitudes and have all negative real parts. Using functional iteration techniques for solving stiff ODE's therefore results in extremely small time steps.

Newton's method, on the other hand, is not so severely restricted. To use Newton's method, Eq. (2.4.3) is re-cast into the fixed point equation[3]

$$y^{(k+1)}_{n+1} = T\left(y^{(k)}_{n+1}\right) := y^{(k)}_{n+1} - \left[F'\left(y^{(k)}_{n+1}\right)\right]^{-1} F\left(y^{(k)}_{n+1}\right) \qquad (2.4.6)$$

where F and its Jacobian F' are defined by

$$F(y_{n+1}) := y_{n+1} - y_n - \frac{\Delta t}{2}\left(f(y_n, t_n) + f(y_{n+1}, t_{n+1})\right) , \qquad (2.4.7)$$

[3] Again Eq. (2.4.6) is called a fixed point equation because a solution, y*, for Eq. (2.4.2) satisfies y* = T(y*)

$$F'(y_{n+1}) = I - \frac{\Delta t}{2} f'(y_{n+1}, t_{n+1}),$$

Here I denotes the identity matrix and f' is the Jacobian of the right hand side of the system of ODE's. In general, Newton's method will converge quadratically to the desired solution if the initial guess $y_{n+1}^{(0)}$ is sufficiently close and if the Jacobian, F', exists, is continuous, and is nonsingular at the solution $y_{n+1}^{(\infty)}$. If all the eigenvalues of f' have real parts less than or equal to 0, then for any positive Δt all the eigenvalues of F' will have real parts greater than or equal to 1, and hence F' will be nonsingular. The functional iteration method, Eq. (2.4.4), on the other hand, only guarantees convergence for step sizes smaller than $2/\|A\|_2$ which can be small for stiff problems. For the model problem y' = Ay, we find the iteration function T for Newton's method to be

$$T\left(y_{n+1}^{(k)}\right) = y_{n+1}^{(k)} - \left[I - \frac{\Delta tA}{2}\right]^{-1}\left\{y_{n+1}^{(k)} - y_n - \frac{\Delta tA}{2}\left(y_{n+1}^{(k)} + y_n\right)\right\} \tag{2.4.8}$$

$$= \left[I - \frac{\Delta tA}{2}\right]^{-1}\left\{I + \frac{\Delta tA}{2}\right\}y_n .$$

For linear ODE's Newton's method converges in one step provided that $I - \Delta tA/2$ is nonsingular.

A disadvantage of Newton's method is that a set of linear equations needs to be solved at each step. An offsetting advantage is that large steps can be taken without running into problems with stability. So even though there is more work per time step (compared to functional iteration), less steps are required which usually means less work to solve the ODE's. A second related disadvantage of Newton's method is that the linear system that must be solved requires $O(N^2)$ storage spaces where

N = number of ordinary differential equations = (4 + 2*NPROD)*NIRM,

NPROD is the number products of combustion including oxygen, and NIRM is the number of inside rooms.

The Jacobian related to the CCFM.VENTS differential equations for a sample problem with three products of combustion has the structure illustrated in Fig 2.4.4.1.2. By structure, it is meant which Jacobian entries are zero or neglible. The presence of many zeros in the Jacobian can be exploited to produce algorithms that are more efficient than those presently used to solve the CCFM.VENTS differential equations.

The terms , '*', '0' or '?' in the Jacobian represents a matrix of size NIRM x NIRM where NIRM is the number of rooms in the simulation. A '*" indicates entries that in general will not be zero. A '?' indicate entries that in general are not zero but seem to be small relative to the diagonal entry for that row and can perhaps be ignored. The '0' indicates entries that are always zero, since products do not interact with each other. The terms Pr y_{lay}, m_l, m_u represent relative pressure, layer height and lower/upper mass. The terms P_il, P_iu for i=1, 2, 3 represent the i'th product of combustion for the lower/upper layer. The i-j'th entry in the Jacobian illustrated in Figure 2.4.4.1.2 represents the $\frac{\partial \text{ i'th equation}}{\partial \text{ j'th variable}}$ where the equations and variable are re-ordered by type (pressure, layer height, mass etc.) rather than by room.

variables

$$
\begin{array}{llcccccccccc}
 & & Pr & y_{lay} & m_l & m_u & P1_l & P1_u & P2_l & P2_u & P3_l & P3_u \\
 & Pr & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 \\
 & y_{lay} & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 \\
 & m_l & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 \\
 & m_u & * & * & * & * & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{equations} & p1_l & ? & ? & ? & ? & * & * & 0 & 0 & 0 & 0 \\
 & p1_u & ? & ? & ? & ? & * & * & 0 & 0 & 0 & 0 \\
 & p2_l & ? & ? & ? & ? & 0 & 0 & * & * & 0 & 0 \\
 & p2_u & ? & ? & ? & ? & 0 & 0 & * & * & 0 & 0 \\
 & p3_l & ? & ? & ? & ? & 0 & 0 & 0 & 0 & * & * \\
 & p3_u & ? & ? & ? & ? & 0 & 0 & 0 & 0 & * & *
\end{array}
$$

Figure 2.4.4.1.2 CCFM.VENTS Jacobian Structure

Making no assumptions about the Jacobian's structure for a 10 room, 10 product case will result in a matrix with

$$(\mathrm{NIRM}*(4+2*\mathrm{NPROD}))^2 = 57600$$

entries, most of which are zero. If the '?' terms can in fact be ignored then the Jacobian for the same case using the structure given in Figure 2.4.4.1.2 will have

$$(4*\mathrm{NIRM})^2 + \mathrm{NPROD}*(2*\mathrm{NIRM})^2 = 5600$$

entries. Computation using fewer elements of the Jacobian will result in an ODE algorithm that is more efficient, i.e. faster, requires less storage and yet be just as accurate as the one presently used in CCFM.VENTS.

2.4.4.2 Implemention Of An ODE Solver In CCFM.VENTS

CCFM.VENTS uses the ODE solver **DEBDF**, which stands for differential equation backward difference formula. An alternate stiff solver using backward difference formulas is **SDRIV3**. The calling sequences of both solvers are similar. They both require the same kinds of information such as error tolerances, starting ending times etc., but the format for the information is slightly different. Each solver is passed the name of a routine, **FBUILD**. This routine calculates the right hand side of the ODE's. **DEBDF** requires that **FBUILD** have the following form:

```
SUBROUTINE FBUILD(TSEC, X, XPRIME, RPAR, IPAR)
```

where TSEC is the independent variable and x is the dependent variable of the ODE. XPRIME is the rhs of the ODE or equivalently the derivative of X with respect to TSEC. RPAR and IPAR are floating point and integer parameter arrays to be passed through to any routines that **FBUILD** calls. Another solver such as **SDRIV3** has different requirements for the rhs calculation routine **FBUILD**. Each place in CCFM.VENTS that calls **FBUILD** would also have to be modified. The routines that call **FBUILD** are listed in Appendix A under called by entry for **FBUILD**. To use **SDRIV3** the form of **FBUILD** would have to be changed to:

```
FBUILD(NEQNS, TSEC, X, XPRIME)
```

where NEQNS are the number differential equations and TSEC, X and XPRIME are the same as before.

2.5 Utility Routines

Utility routines are those subroutines/functions in CCFM.VENTS that support some computational aspect of the various software modules. For the most part they can exist outside of CCFM.VENTS. Some examples of utility routines present in CCFM.VENTS are:

- **MACH routines** - Standard subroutines: **I1MACH, R1MACH, D1MACH**, originally developed at Bell Laboratories for recording the floating point characteristics of a computer

- **Memory management routines** - A collection of subroutines written in standard Fortran used for allocating and de-allocating memory.

- **Character handling** - A collection of subroutines for converting character data to floating point, integer or logical data types.

- **Message printing** - A subroutine for printing messages and various levels of error messages

- **Numerical Routines** - routines for sorting, solving ODE's, etc.

The sorting routines were modified for use with the vent routines and so are not useful outside of CCFM.

2.5.1 Computer Floating Point Characteristics

The computer floating point characteristic of most interest is the unit rounding or machine unit or commonly referred to as machine epsilon which is defined to be the smallest positive floating point number which can be added to one to produce a machine number different from one. This machine dependent parameter depends on the length of the fractional portion (mantissa) of the machine representation for a real number.

A standard approach to obtain the correct floating point parameters is to put the parameters for all commercial machines into data statements in function subprograms with the names I1MACH, for integer parameters, R1MACH, for real parameters, and D1MACH, for double precision parameters. When a code such as CCFM.VENTS is installed on a new machine, the installer must go into these three routines and activate the data statements corresponding to his machine. This involves changing a few comment statements into lines of code.

Four key floating point characteristics of a computer are: the base of the floating point representation, β; the number of digits in the mantissa, t; the smallest exponent, e_{min}; and the largest exponent, e_{max}. A set of numbers can be generated from these four parameters using the following floating point representation.

$$(\beta_1 \times \beta^0 + \beta_2 \times \beta^{-1} + \ldots + \beta_t \times \beta^{1-t}) \times \beta^e \tag{2.5.1}$$

where

$1 \leq \beta_1 \leq \beta - 1$

$0 \leq \beta_i \leq \beta - 1;$ $\quad i = 2, \ldots$ or t

$e_{min} \leq e \leq e_{max}.$

The numbers: $\beta_1, \ldots, \beta_t$ form the mantissa and e forms the exponent of the floating point number used on the computer. The set of numbers defined by Eq. (2.5.1) is called the Wilkinson Floating Point Model. One particular parameter of interest, machine epsilon, can be derived from β and t. Consider the number 1 written in the format of Eq. (2.5.1), i.e. $\beta_1 = 1$, $\beta_2 = \ldots = \beta_t = 0$ and e = 0. The smallest number that can added to 1 is a number where the exponent and the first t-1 digits of the mantissa are zero and the t'th digit is 1. This number is β^{1-t} or machine epsilon. The significance of machine epsilon is that it indicates how close two machine numbers may be before catastrophic cancellation (loss of all leading significant digits) can occur. Floating point parameters for a few computers of interest are given Table 2.5.1.

Table 2.5.1 Floating Point Characteristics for Various Computers

Computer	β	t	e_{min}	e_{max}	$\varepsilon_m = \beta^{1-t}$
Univac 1100	2	27	-128	127	1.49×10^{-8}
PDP-11/Vax 11/780	2	24	-127	127	1.19×10^{-7}
Cray - 1	2	47	-8189	8190	1.42×10^{-14}
Apple Macintosh/ IBM PC	2	24	-125	127	1.19×10^{-7}
IBM 360/370/ Concurrent 7/32	16	6	-64	63	9.54×10^{-7}
CDC 200 (Cyber 205)	2	47	-28625	28718	1.42×10^{-14}

β = base, t = number of digits in mantissa
e_{min} = smallest exponent, e_{max} = largest exponent
ε_m = machine epsilon; smallest floating point number such that $1 + \varepsilon_m > 1$

2.5.2 Memory Management Routines

Memory management routines are a collection of routines written in standard Fortran that provide a means for allocating and de-allocating memory. These allocation/de-allocation routines are similar to the capabilities provided with C and Pascal. CCFM.VENTS's allocation process is not truly dynamic, for all memory assigned by the memory management routines comes from the common block /**MEMRY**/. This block of memory can only be re-sized at compile time. The present version of CCFM.VENTS allocates 2000 floating point words to this common block.

The fundamental unit of allocated memory is called a block. The structure of a memory block is given in Figure 2.5.2.1. There is a storage overhead of 5 integer units per block. This extra memory is used to store pointers to the next block, the previous block, the size of the block, the type of block (integer or floating point) and an error detection code. The error detection code is a heuristic for determining whether a pointer passed to a memory management routine represents a valid block of memory. So the 5'th unit of overhead is chosen so that all overhead units for a given block sum to 10. So whenever a memory management routine is called it sums the overhead block. If the sum is not 10 it assumes that it was called with an invalid memory pointer and returns an error code.

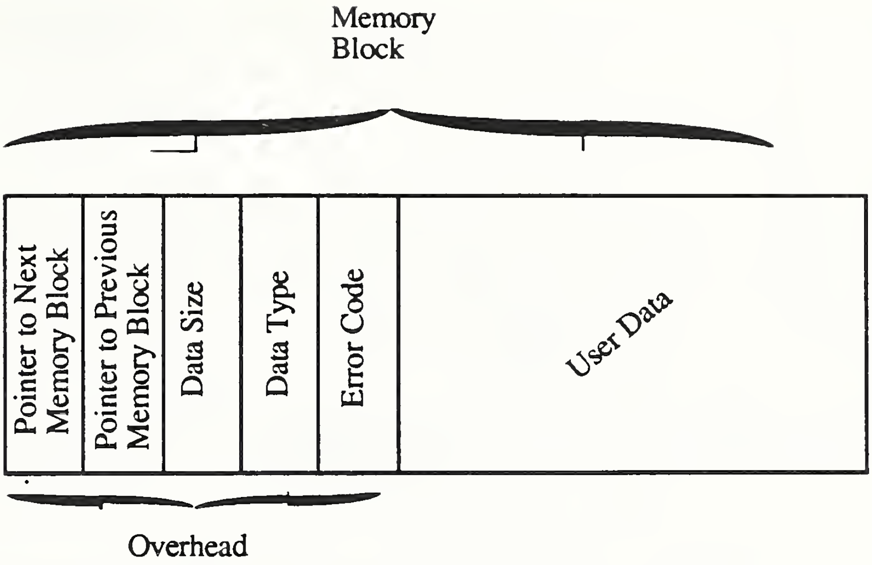

Memory Block
Pointer to Next Memory Block
Pointer to Previous Memory Block
Data Size
Data Type
Error Code
User Data
Overhead

Figure 2.5.2.1 Memory Block Data Format

The operations presently supported by with the memory management routines are:

- **GETPTR** - allocate a block of memory
- **DELPTR** - de-allocate a block of memory
- **CPYPTR** - make a copy of a block of memory
- **RESIZ** - change the space allocated to a memory block
- **SIZE** - return the size of a memory type. The size is in terms of the memory type, integer or floating point, of the allocated block.

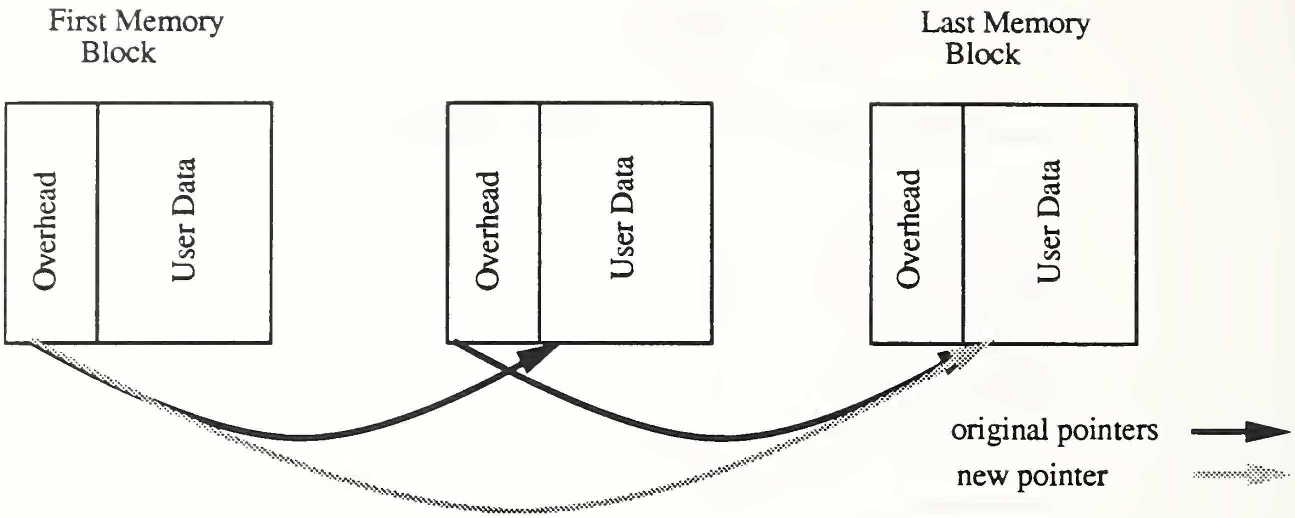


Figure 2.5.2.2 Deleting a Memory Block

Memory blocks are connected to each other with doubly-linked lists. The forward links are shown in Figure 2.5.2.2. To allocate a block of memory, **GETPTR**, simply walks down the links as illustrated by the arrows until the free space after a block is sufficient to contain a block of the requested size. **GETPTR** then installs this block in the chain by re-establishing the links. Likewise to delete a memory block **DELPTR** simply cuts the links referenced by the deleted block. The solid arrows represent forward links before a deletion, shaded arrows represent links after a deletion. A similar operation must be performed for the backward links.

2.5.3 Character Handling Routines

There are three routines for converting character data to "value" data where "value" refers to floating point numbers, integers or logicals. These routines are RVAL/DVAL, INTVAL and LVAL.

2.5.4 Error/Message Printing

The error message/printing routines provide a central location within CCFM.VENTS for communicating with the user. There are times when it is desirable to turn off the routines display of user messages. For example, when the user is LOADING a data set it is not necessary to see all the messages generated by each command in the file. By having all of the output occur in one subroutine it is convenient to turn it off as in the previous example. This message level displayed can then be controlled by the user.

The user/developer uses this routine by passing it a character string containing the message or error to be printed and the level of the message. The message will be printed as long as its level exceeds the minimum message priority as defined by the variable **IOLEVL**. The message priority levels are

- **informative** - A level 0 message, just communicates information to the user by displaying model results, for example. There is no connotation of an error or problem with this type of message.

- **warning** - A level 1 message, indicates a possible problem that may or may not require corrective action.

- **error** - A level 2 message, indicates a situation requiring some action to remedy. Further results are suspect unless the problem is corrected. Example - an input command is entered incorrectly

- **fatal error** - A level 3 message indicates a problem has been encountered where no conceivable action can be taken to remedy the situation. Example - division by 0.

3. Developing CCFM

3.1 Adding New Physical Routines

The place in CCFM.VENTS where it is logical to install additional physical algorithms is within the subroutine **FBUILD**. **FBUILD** is responsible for calculating the ODE rhs. The ODE rhs depends upon contributions to layer mass, enthalpy and product of combustion flows due to each phenomena being modeled. After calculating these contributions **FBUILD** totals them, and then calculates the ODE rhs. The following code segment from **FBUILD** shows where a new routine, **XPHEN**, describing a new phenomenon would be called and how the totaling code would be modified.

FBUILD first calculates the mass, enthalpy and product of combustion flows for each physical phenomenon. Presently these phenomenon are natural vents - **UVENT**, forced vents - **FVENT** and fire plumes - **FPLUME**. These routines return the flows in the data structures **FLWU**, **FLWF** and **FLWP** respectively. These structures are each organized as illustrated in Figure 2.4.1.3. After the flows are calculated they are summed. This sum is stored in **FLWTOT** which again is organized in the same way as **FLWU**, **FLWF** and **FLWP**. Finally, **FROOM** is called to calculate the ODE rhs. The changes necessary to implement the new fire phenomenon **XPHEN** are highlighed in **bold**.

```
C
C*** CALCULATE FLOW'S DUE TO NATURAL VENTS
C
      CALL UVENT(TSEC,X,FLWU)
C
C*** CALCULATE FLOW'S DUE TO FORCED VENTS
C
      CALL FVENT(TSEC,X,FLWF)
C
C*** CALCULATE FLOW'S DUE TO PLUMES
C
      CALL FPLUME(TSEC,X,FLWP,QLPQUP)
C
C***  CALCULATE  FLOW  FOR  NEW  PHENOMENA
C
          CALL  XPHEN(TSEC,X,FLWX)
C
C*** SUM ALL FLOW'S
C
      DO 20 JROOM= 1, NIRM
         QLPQUR(JROOM) = FLWU(JROOM,Q,L)+FLWU(JROOM,Q,U) + QLPQUP(JROOM)
```

```
     *                 + FLWF(JROOM,Q,L)+FLWF(JROOM,Q,U)
      *                  +  FLWX(JROOM,Q,L)+FLWX(JROOM,Q,U)
         DO 21 ILAY = 1, 2
           DO 22 IPROD = 1, NPROD+2
               FLWTOT(JROOM,IPROD,ILAY) = FLWU(JROOM,IPROD,ILAY) +
     *            FLWP(JROOM,IPROD,ILAY) + FLWF(JROOM,IPROD,ILAY)
      *           + FLWX(JROOM,IPROD,ILAY)
   22       CONTINUE
   21    CONTINUE
   20 CONTINUE

C
C*** CALCULATE RHS OF ODE'S FOR EACH ROOM
C
      DO 10 I = 1, NIRM
         JROOM = I
         II = 1 + (JROOM-1)*NRMDE
         CALL FROOM(
     I        TSEC, X(II), FLWTOT,QLPQUR,JROOM,
     O        XPRIME(II)
     +              )
   10 CONTINUE
```

3.2 Adding or Changing Menu commands

As illustrated in the previous code segment it is relatively easy to install additional physical algorithms to CCFM.VENTS. The hard part then is to write the routines in the first place and to make the necessary changes in CCFM.VENTS to solicit any additional inputs from the user required by the new algorithm. The new input requirements will more than likely require changes in the subroutine **MENU1** and additions or modifications to CCFM.VENTS's global data structures (Common Blocks). Algorithm 3.2.1 gives a brief synopsis of the work performed by MENU1 when executing a menu command.

Algorithm 3.2.1 MENU1 Calculations

1. Branch to the section of **MENU1** for handling the entered CCFM.VENTS command as indicated by the argument IPROG. This argument is known as the command reference number (CRN) . A list of CRN's is given in Table 3.2.1. These values are read in by CCFM.VENTS from the menu definition file.
2. Handle the command. A typical CCFM.VENTS command, **COM1**, is handled in the following way.
   - 2.1 Determine whether data is being entered using screen mode. If screen mode is desired then construct a command line of the current data for **COM1**. Pass this command line to the routine screen handling routine **SCRNIN**.
   - 2.2 If screen mode was used then use value of **COM1** passed back by **SCRNIN** in the following steps, otherwise use the value of **COM1** originally entered by the user.
   - 2.3 Convert the tokens on **COM1** from character data to the appropriate type and check whether the converted data falls within the required bounds. This work is done with the routines: **IBNCHK** for

```
        integer conversions; RBNCHK for floating point conversions; LBNCHK for logical conversions;
        and FBNCHK for file checking.
2.4     Transfer numerical data from tokens to CCFM.VENTS global data structures.
2.5     Call the routine MSGPRT to print out data that was entered.  Note: this message display can be
        turned off by setting the flag IOLEVL to 1.
```

A command reference number (CRN) is the index of a **COMPUTED GOTO** statement in **MENU1** or **MENU0** that corresponds to the command being executed. The CRN's for CCFM.VENTS are given in Table 3.2.1. These values are given in the menu definition file that CCFM.VENTS reads in to define the menu's. The command **AMB** has a CRN of 5. Inside the the menu definition file there is the following statement:

```
.C      AMB   5
```

Whenever the **AMB** command is executed a value of 5 is passed to **MENU1** in the variable **IPROG**. The COMPUTED GOTO statement within MENU1 then branches to the statement

```
170  CONTINUE
```

By placing the COMPUTED GOTO index in a data file for each command the commands may be rearranged arbitrarily without changing or re-compiling any subroutines. A new CCFM.VENTS command may be installed by following the procedure outlined in Algorithm 3.2.2.

Table 3.2.1 Command Reference Numbers (CRN)[4]

Command	CRN	Command	CRN
AMB	5	AUTO	18
BEGIN	16	CONS	1
CONV	14	DUMP	30
FACTOR	8	FILES	22
FIRE	7	FVENT	9
HELP	-7	INIT	17
INITP	6	LOAD	29
NUM	4	NUMINF	12
OPTIONS	-6	PARMS	13
PATH	2	QUIT	-5
REPORT	27	RERUN	28
ROOM	11	TIME	15
TITLE	25	VALUES	31
VENT	10		

[4] Commands corresponding to negative CRN's are generic commands and are executed by the subroutine **MENU0**.

Algorithm 3.2.2 Installing a New Menu Command

1. Suppose the new command is named **NEWC.**
2. Pick a unique command reference number for **NEWC**. CRN's already taken are given in Table 3.3.1. CRN's of 3 and 9 are available so let **NEWC**'s CRN be 3.
3. Add the following text to the menu definition file. Place this text immediately after the command you wish NEWC to follow in CCFM.VENTS's menu.

```
.C      NEWC   3
```

4. Place any help text or parameter description text after the entry above.

```
.H      Any help text for the new command NEWC
.H      Some more help text for the new command NEWC
.PD     description of NEWC's first parameter
.PD     description of NEWC's second parameter, etc.
```

5. Modify **MENU1.** Examine the computed goto statement in MENU1. The label that will be executed when IPROG=3 is

```
        150 CONTINUE
```

   5.1 Place code to implement the command **NEWC** after the statement, 150 CONTINUE.

   5.2 Use Algorithm 3.2.2 as a model for executing the command **NEWC.** Use the routines RBNCHK, IBNCHK, LBNCHK and/or FBNCHK to copy information from the character string CARD that was entered by the user to the global data structures that are associated with this command.

6. Add an entry to the subroutine **DUMP**. Each CCFM.VENTS command has a corresponding entry in the subroutine **DUMP**. **DUMP** simply does the opposite of **MENU1**. **MENU1** transfers data found on the command line to the appropriate global data structures. **DUMP** takes data from the global data structures and constructs a command line.

3.3 Installing CCFM.VENTS on an IBM-PC Compatible Computer

CCFM.VENTS is designed to run on any IBM-PC compatible computer with 640k of memory that has a floating point chip installed. This chip is necessary in order to run CCFM.VENTS in a timely manner, since CCFM.VENTS is floating point intensive. CCFM.VENTS should be installed in its own directory. Copy all files on the distribution floppy disk to a directory on the hard disk. To test if CCFM.VENTS was installed correctly change to that directory and type:

CCFM

followed by

BEGIN.

Results similar to those printed on your terminal screen should appear in section 2 of Part IV [12] of this report, the user's reference guide.

The installation is complete.

3.4 Porting CCFM.VENTS to a non-IBM-PC Computer

The CCFM.VENTS code which is accessible from the CFR computer bulletin board, will run on an IBM-PC or compatible or an Apple Macintosh. The following notes indicate what parts of CCFM.VENTS's source code may need to be changed in order to install it on another computer.

The only critical step in installing CCFM.VENTS onto another computer is setting up the Fortran I/O unit numbers correctly and installing the correct MACH routine machine parameters. The two other steps: accessing the CPU clock and defining the file checking routine LEGAL can be ignored if you do not wish to use the features that they provide.

Setting Fortran I/O units The unit numbers and file names used for reading and writing data to the terminal screen are defined in the main program. In general the unit numbers and file names used to refer the "terminal screen" are different for each computer.

The following code segment was taken from the main routine of CCFM.VENTS. The variables containing the unit numbers for input and output are IN1 and IOUT1 respectively. Most FORTRAN's assign 5 for terminal input and 6 for terminal output. This is not universal, however. Again, the values used for FORTRAN unit numbers for screen I/O are not defined by the standard.

```
IOUT2 = 9
IOUTER = 9
IF(COMPTR.EQ.NOS)THEN
     ITERMI = 5                                  ; CYBER/NOS
     ITERMO = 6
     ISCRAT = 9
     OPEN(UNIT=ITERMI,FILE='INPUT')
     OPEN(UNIT=ITERMO,FILE='OUTPUT')
    ELSE IF(COMPTR.EQ.MAC)THEN
     ITERMI = 9                                  ; Macintosh
     ITERMO = 9
     ISCRAT = 12
    ELSE IF(COMPTR.EQ.CON)THEN
     ITERMI = 5                                  ; Concurrent 3252
     ITERMO = 6
     ISCRAT = 9
```

```
         OPEN(UNIT=ITERMI,FILE='C:')
         OPEN(UNIT=ITERMO,FILE='C:')
        ELSE IF(COMPTR.EQ.IBMPC.OR.COMPTR.EQ.COMPAQ)THEN
         ITERMI = 5                                ; IBMPC Compatible
         ITERMO = 6
         ISCRAT = 9
        ELSE IF(COMPTR.EQ.UNIX)THEN
         ITERMI = 5                                ; UNIX Work Station
         ITERMO = 6
         ISCRAT = 9
        ELSE
         ITERMI = 5
         ITERMO = 6
         ISCRAT = 9
         OPEN(UNIT=ITERMI,FILE='INPUT')
         OPEN(UNIT=ITERMO,FILE='OUTPUT')
      ENDIF
```

Setting the MACH routines The MACH routines, R1MACH, D1MACH and I1MACH were originally written at Bell Laboratories as a part of their PORT library. They are in the public domain. These routines are used to specify the floating point environment upon which the computer program that is using them is running. The source for these routines contains commented examples for a large number of commonly used computers, ranging from a Cray 1 to an IBM PC or Macintosh. To change from the IBM-PC to another computer say a VAX-11/780 you simply remove a 'C' from column 1 of the data statements for the VAX-11/780 and place a 'C' in column 1 for each data statement referring to the IBM-PC. This needs to be done for I1MACH. It is also required for R1MACH if you are going to use single precision and D1MACH otherwise. A portion of I1MACH is given below for the IBM PC Family and the Vax 11-780.

```
C     MACHINE CONSTANTS FOR THE IBM PC FAMILY (D. KAHANER NBS)
C
C     DATA IMACH/5,6,0,6,32,4,2,31,2147483647,2,24,
C     * -125,127,53,-1021,1023/
C               NOTE! I1MACH(3) IS NOT WELL DEFINED AND IS SET TO ZERO.


C     MACHINE CONSTANTS FOR THE VAX 11/780
C
C     DATA IMACH(1) /    5 /
C     DATA IMACH(2) /    6 /
C     DATA IMACH(3) /    5 /
C     DATA IMACH(4) /    6 /
C     DATA IMACH(5) /   32 /
C     DATA IMACH(6) /    4 /
C     DATA IMACH(7) /    2 /
C     DATA IMACH(8) /   31 /
C     DATA IMACH(9) /2147483647 /
```

```
C     DATA IMACH(10)/    2 /
C     DATA IMACH(11)/   24 /
C     DATA IMACH(12)/ -127 /
C     DATA IMACH(13)/  127 /
C     DATA IMACH(14)/   56 /
C     DATA IMACH(15)/ -127 /
C     DATA IMACH(16)/  127 /
C
```

Accessing the CPU clock CCFM.VENTS calls a routine named CPTIME any time it wants the elapsed CPU time in seconds since the beginning of the run. CCFM.VENTS uses this information to calculate solver times so that the user can know how hard the program is working to solve a problem. The code segment listed below gives the way that three different computers calculate elapsed CPU time. If you do not desire this feature then simply set CPUTIM to 0. The information on how to access the CPU clock is usually given in your computer system FORTRAN manual. Access of the CPU clock is not a feature of FORTRAN 77.

The fundamental unit of time on a Macintosh is called a tick and is about 1/60th of a second. The value in ticks since the Macintosh was turned on is located at memory location: 16A hex or 362 decimal. In order to return a value in seconds the number of ticks must be scaled by 60. The PC has a routine named TIMER that returns the number of ticks. This is similar to a Mac except that 1 tick on a PC is 1/100'th of a second rather than 1/60'th.

```
      SUBROUTINE CPTIME(CPUTIM)
                      ●
                      ●
                      ●
C
C*** CYBER/NOS
C
C     CPUTIM = SECOND()
C
C*** MACINTOSH/ABSOFT FORTRAN
C
C     CPUTIM = LONG(Z'16A')/60.0D0
C
C*** IBM-PC COMPATIBLE, MS-DOS - LAHEY FORTRAN
C
      CALL TIMER(ITICK)
      CPUTIM = ITICK/100.0D0
C
C*** SET CPUTIM TO 0. IF YOU DON'T KNOW TO GET CPU TIME
C     CPUTIM = 0.0D0
      RETURN
      END
```

Defining the File Checking Routine A function subroutine named **LEGAL** has been written to check the validity of file names. Since file naming conventions are functions of the computers operating system it is necessary to have a different version of **LEGAL** for each computer where CCFM.VENTS is installed. **LEGAL** is passed a character string that contains the name of a file. **LEGAL** then passes back to the calling routine a true value if the file name was legal and a false value otherwise. This routine is called by any CCFM.VENTS command that uses file names such as **FILES**, **DUMP** and **LOAD**. To quickly install CCFM.VENTS on a new computer you may simply always return true. The following code segment from the function subroutine LEGAL shows how file error checking was implemented on the Cyber. The Cyber requires file names have length no longer that 7 characters. The additional requirement was added that the first character had to be alphabetic and each subsequent character had to be a digit or alphabetic.

```
   IF(COMPTR.EQ.NOS)THEN
      LEGAL = .FALSE.
      N1 = LENGTH(FILE)
      IF(N1.LE.0)THEN
         CALL MSGPRT('FILE NAME TO SHORT',2)
         RETURN
      ENDIF
      IF(N1.GT.7)THEN
         CALL MSGPRT('FILE NAME TO LONG',2)
         RETURN
      ENDIF
      IF(.NOT.ALPHA(FILE(1:1)))THEN
         WRITE(MSSG,11)FILE(1:1)
         CALL MSGPRT(MSSG,2)
         RETURN
      ENDIF
      M = 7
      IF(N1.LT.7)M = N1
      DO 10 I = 2, M
         C = FILE(I:I)
         IF(ALPHA(C))GO TO 10
         IF(NUM(C))GO TO 10
         IF(C.EQ.' ')GO TO 10
         WRITE(MSSG,11)C
11       FORMAT(' ILLEGAL CHARACTER:',A1,':')
         CALL MSGPRT(MSSG,1,1,2)
         RETURN
10    CONTINUE
      LEGAL = .TRUE.
     ELSE
      LEGAL = .TRUE.
   ENDIF
```

REFERENCES

[1] ANSI X3.9-1978, American National Standard Programming Language FORTRAN, American National Standards Institute, 1430 Broadway, New York, New York 10018

[2] Numerical Recipes, The Art of Scientific Computing, Editors: William H. Press, Brian P. Flannery, Saul A. Teukolsky and William T. Vetterling, Camebridge University Press

[3] Shampine, L. F. and Gear, C. W., A User's View of Solving Stiff Ordinary Differential Equations, SIAM Review V21, p.1-17. (1979)

[4] Byrne, George D. and Hindmarsh, Alan C., Stiff ODE Solvers: A Review of Current and Coming Attractions, Journal of Computational Physics V70, p. 1-62. (1987)

[5] Gear, C.W., The Automatic Integration of Stiff Ordinary Differential Equations, Information Processing p. 187-193. (1969)

[6] Gear, C.W., Algorithm 407: DIFSUB for solution of ordinary differential equations [D2], Comm. ACM V14, p. 185-190. (1971)

[7] Watts, H. A. and Shampine, L. F., BIT V12, p.252- (1972)

[8] William F. Moss, private communication

[9] Introductory Functional Analysis With Applications, Erwin Kreyszig, John Wiley & Sons, New York (1978)

[10] Cooper, L.Y. and Forney, G.P., The Consolidated Compartment Fire Model (CCFM) Computer Code Application CCFM.VENTS - Part I: Physical Basis, NISTIR 90-4342, National Institute of Standards and Technology, Gaithersburg MD, 1990.

[11] Forney, G.P. and Cooper, L.Y., Eds., The Consolidated Compartment Fire Model (CCFM) Computer Code Application CCFM.VENTS - Part III: Catalog of Algorithms and Subroutines, NISTIR 90-4344, National Institute of Standards and Technology, Gaithersburg MD, 1990.

[12] Forney, G.P., Cooper, L.Y. and Moss, W.F., The Consolidated Compartment Fire Model (CCFM) Computer Code Application CCFM.VENTS - Part IV: User Reference Guide, NISTIR 90-4345, National Institute of Standards and Technology, Gaithersburg MD, 1990.

APPENDICES

A. Glossary of CCFM.VENT Subroutines

This appendix contains a brief description of the the routines found in CCFM.VENT arranged by functional type. Each routine description begins with the term *generic* or *specific*. Routines that can be used "as is" in other computer programs are denoted *generic*. These routines do not use or depend upon data structures specific to CCFM.VENTS. All others routines are denoted *specific.*

This appendix also contains information on how each subroutine and common block in CCFM.VENTS are related to each other. This information is essential to the person who wishes to modify CCFM.VENTS. For example, if one wishes to change the number of arguments in the calling list of a subroutine one would need to know what subroutines call it.

The main heading, **NAME**, is given in **bold text.** Each entry has a description and up to four sub-headings: CALLS, LIB,COMMONS and CALLED BY. The subheadings CALLS and LIB are similar. They both list external references to **NAME.**, i.e. what routines **NAME** calls. The source code for the routines that are listed under CALLS appear in the same computer file as **NAME**. On the other hand, the routines listed by the LIB sub-heading do not. Some examples of routines that would appear under LIB are FORTRAN supplied functions such as ABS, SQRT, MOD, etc. The names listed next to COMMONS are the COMMON blocks that appear in the routine **NAME**. The routines that are listed next to CALLED BY are those routine that call **NAME.**

NAME ... description of NAME ...

CALLS: SUB1, SUB2, ...
LIB: SUBA, SUBB, ...
COMMONS: COM1, COM2, ...
CALLED BY: SUBa, SUBb, ...

A.1 INPUT Module Routines

ARAYIN *generic* - The driver routine for the array input package. It provides a facility for inputing data that is organized as vectors or arrays. The vector data is stored using the memory manager (see utility routines). So the only limit on array size is the memory available.

CALLS: ARFRSH COLAPS CPYPTR DCODE DELPTR DELVAL GETHLP GETITM GETPTR GETSIZ IBNCHK INCLST INTVAL MSGPRT PARSE RESIZ RVAL SORT
LIB: IABS MAX MIN
COMMONS: FIO MEMRY MEMRY2 MENU2 MENU4 MENUS MFLAGS MIO TOKEN1 TOKEN2
CALLED BY: ARYTOK

ARYTOK *generic* - Identify the itok'th token on a command line as either a floating point or an array/vector flag. An array/vector flag is indicated by a string of characters beginning with 'V'. The subroutine arguments, ixptr, iyptr, xlabl, ylabl, are the same as defined in the routine **ARAYIN**.

CALLS: MSGPRT
CALLED BY: ARAYIN

INCLST *generic* - This routine is part of the array input package. This package will display up to 10 elements of an array. This routine determines which element should be the last one displayed.

CALLED BY: ARAYIN

DELVAL *generic* - A subroutine used in the array input package. This routine marks data as deleted that lie between a given lower (xmin) and upper (xmax) bound. Delval then sets the corresponding elements in the array ISET to 0; to indicate that these elements have been deleted.

CALLED BY: ARAYIN

COLAPS *generic* - A subroutine used in the array input package. While in the array input mode the user can delete array data values that lie between a given lower and upper bound. Internally the data is marked as deleted using the array ISET. This routine recovers the un-used space by collapsing the arrays containing the deleted data.

CALLED BY: ARAYIN

SCRCMD *generic* - A subroutine used in the array input package and the screen input package. It is used to convert character strings, 'h', 'o', 'c', etc to integers. These characters refer to the screen and array input commands: help, ok and cancel.

ARFRSH *generic* - A subroutine used in the array input package. It refreshes the screen in the array input mode. The user can modify and/or delete data by field. To see the results of data editing, the user types a null return or d for display. This routine is then called to print out an updated screen of data.

CALLS: MSGPRT
CALLED BY: ARAYIN

DUMP *specific* - Writes the CCFM.VENTS commands that specify the current case being modeled to an output file. The output file has a unit number, iodump. If IWHICH=1 then all CCFM.VENTS commands are dumped. If IWHICH=0 then only those commands that have changed since the last time step are dumped.

CALLS: GENCHK GETSIZ LENGTH RMVBLK
COMMONS: CHECK CONST ERCOM ERRCHR FILES FIRE FLAGS
MEMRY MEMRY2 MFLAGS COMMONS: MODCOM OUTS
PRODS ROOM SIZES TITLS VARMAP VENTS
CALLED BY: MENU1 MODEL

FBNCHK *generic* - Checks whether a token on an input line can be opened as a file for either input or output.

CALLS: LEGAL MSGPRT
COMMONS: TOKEN1 TOKEN2
CALLED BY: MENU1

GENCHK *specific* - This routine checks for certain types of generic input errors. two types of errors that are checked are: defining a product of combustion but not defining a factor or defining a vent (either natural or forced) that references a room but not defining that room.

CALLS: NUMINF
COMMONS: CHECK SIZES
CALLED BY: DUMP MENU1

GETHLP *generic* - Displays help information about a command. The routine GETITM associates an identification number, inum, with each command. This routine uses inum as an index to retrieve help information about a command.

CALLS: MSGPRT
LIB: MOD
COMMONS: COMTYP FIO MENU2 MENU4 MENUS MIO

GETITM *generic* - This routine determines whether a given token is a valid command and if so returns an identification number. This information is used by the routines MENU0 or MENU1 to execute the command and by GETHLP to display help information about the command.

CALLS: UPPER
COMMONS: MENU2 MENU4 MENUS
CALLED BY: ARAYIN INPUT MENU0 SCRNIN

GETVCT generic - A low-level routine in the array-input module. It is used to store a message into a character string depending on the size of a given dynamic array defined by the memory manager. A blank is stored if the array is of size 0, the array's value is stored if it is of size 1 and the character string 'vector' is stored if the array is of size greater than one.

CALLS: GETSIZ
COMMONS: MEMRY MEMRY2
CALLED BY: MENU1

IBNCHK *generic* - Checks whether a given input token is a valid integer. If it is, it further checks whether the integer is within specified bounds. The following parameter statement is used to determine the type of bounds desired

> PARAMETER (LE=1,LT=2,GE=1,GT=2,NIL=0)
> where LE, LT, GE and GT have the standard FORTRAN meaning
> and NIL means that no bound check is desired

CALLS: INTVAL MSGPRT
COMMONS: TOKEN1 TOKEN2

CALLED BY: ARAYIN MENU0 MENU1 SCRNIN

INPUT *generic* - Driver routine for the INPUT package; it reads in a command line, and causes the line to be interpreted and executed.

CALLS: GETITM MENU0 MENU1 MSGPRT PARSE PRTMEN RDMENU
COMMONS: ERCOM ERRCHR FIO MENU2 MENU4 MENUS MFLAGS MIO TOKEN1 TOKEN2
CALLED BY: MAIN

LBNCHK *generic* - Checks whether a given input token is a valid boolean flag (yes, on, true or no, off, false). This routine uses arrays defined by the routine CPYVAL.

CALLS: LVAL MSGPRT
COMMONS: TOKEN1 TOKEN2
CALLED BY: MENU0 MENU1

MENU0 *generic* Implements the generic commands: BATCH, BRIEF, HELP, LEVEL, STOP and TTY. Any generic commands that are implemented in the future should be handled by this routine.

CALLS: GETHLP GETITM IBNCHK LBNCHK MSGPRT SCRNIN
LIB: IABS
COMMONS: ERCOM ERRCHR FIO MFLAGS MIO TOKEN1 TOKEN2
CALLED BY: INPUT

MENU1 *specific*This routine interprets and acts on the commands entered by the user and are specific to the program CCFM.VENT. This routine and the routine DUMP are the only routines in the INPUT package that need be modified when the input requirements change, i.e a new command is added to the menu.

CALLS: ARYTOK DUMP FBNCHK GENCHK GETVCT IBNCHK INITMD LBNCHK LENGTH MSGPRT NUMINF OPENFL OUTPUT OUTWRP RBNCHK RMVBLK SCRNIN SETDE STATE1 UPPER
COMMONS: CHECK CONST ERCOM ERRCHR FILES FIO FIRE FLAGS INITS IO MEMRY MEMRY2 MFLAGS MIO MODCOM OUTS PRODS ROOM SIZES TITLS TOKEN1 TOKEN2 VENTS
CALLED BY: INPUT

PARSE *generic* - Given a line of text, this routine determines the beginning and ending of each contiguous set of non-blank characters. Other routines in the INPUT module use this information to determine the location and value of parameters found on a command line.

CALLS: GTFCHR INITLV LSTCHR NXTQTE

LIB: LEN
CALLED BY: ARAYIN INPUT RDMENU SCRNIN

PRTMEN *generic* - Prints out the menu as specified by the file MENUMLT.

CALLS: MSGPRT
COMMONS: MENU2 MENU4 MENUS
CALLED BY: INPUT

RBNCHK *generic* - Checks whether a given input token is a valid floating point number. if it is, it further checks whether the floating point number is within specified bounds. The following parameter statement is used to determine the type of bounds desired

```
PARAMETER (LE=1,LT=2,GE=1,GT=2,NIL=0)
```

where LE, LT, GE and GT have the standard FORTRAN meaning and nil means that no bound check is desired

CALLS: MSGPRT RVAL
COMMONS: TOKEN1 TOKEN2
CALLED BY: MENU1

RDMENU *generic* - This routine reads in a text file that describes the structure of the menu. Changes made to individual menu items can be made without re-compiling the program. RDMENU also loads CCFM.VENT with a default data values.

CALLS: INDEXX INTVAL MSGPRT PARSE
LIB: INDEX
COMMONS: FIO MENU2 MENU4 MENUS MFLAGS MIO TOKEN1 TOKEN2
CALLED BY: INPUT

RFRSH *generic* - A routine used while in the screen input mode. Used with the routine SCRNIN to refresh the screen of input values and descriptions.

CALLS: MSGPRT
COMMONS: FIO IO MENU2 MENU4 MENUS MIO
CALLED BY: SCRNIN

SCRNIN *generic* - The driver routine for the screen-input module. The routines implements the screen input mode by soliciting input from the user using a pseudo-screen mode. A list of input descriptions with their corresponding values are printed along with available options. The user can change any input field by typing the field number along with the new value.

CALLS: CRDMRG GETHLP GETITM IBNCHK MSGPRT PARSE RFRSH
LIB: IABS

COMMONS: FIO MENU2 MENU4 MENUS MFLAGS MIO TOKEN1 TOKEN2
CALLED BY: MENU0 MENU1

A.2 MODEL/PHYSICS Module Routines

COMWL1 *generic* - setup for calculation of the flow through vents in the wall segment common to two rooms. this routine calculates room pressures and room-to-room pressure differences at certain elevations along common walls of adjacent rooms. these elevations are:

1. ymin - maximum of two rooms floor elevations
2. ymax - minimum of two rooms ceiling elevations
3. layer elevations between ymin and ymax
4. neutral planes elevations between ymin and ymax

There can be up to 7 elevations of interest (1 floor, 1 ceiling, 2 layer and 3 neutral plane elevations).

CALLS: DELP HSORT RMVDUP
LIB: MAX MIN
CALLED BY: UVENT

DELP *generic* - Calculation of the absolute hydrostatic pressures at a specified elevation in each of two adjacent rooms and the pressure difference. The basic calculation involves a determination and differencing of hydrostatic pressures above a specified datum pressure.

CALLED BY: COMWL1 VENTHP

FANRES *generic* - Find the mass that a fan will deliver under given conditions in the inlet and outlet rooms.

CALLS: FANFOR FANREV INTERP MRES RTSAF2
CALLED BY: VENTF

FANFOR *generic* - This routine is called by a zero finder to locate a pressure difference accross the fan, DPFAN, that will cause this function to be zero when the flow is in the forward or normal direction.

CALLS: INTERP MRES
CALLED BY: FANRES

FANREV *generic* - This routine is called by a zero finder to locate a pressure difference accross the fan, DPFAN, that will cause this function to be zero when the flow is in the reverse direction.

CALLS: INTERP MRES
CALLED BY: FANRES

FBUILD *specific* - Calculates the right hand side of the ODE being solved. The calling sequence of FBUILD is determined by the specific ode solver that is used.

CALLS: CPTIME FPLUME FROOM FVENT SETDE UVENT
COMMONS: FLAGS FLOWS IO MODCOM SIZES
CALLED BY: EIGF MODEL OUTPUT

FLOGO2 *generic* - Deposition of mass, enthalpy, oxygen, and other product-of-combustion flows passing between two rooms through a vertical, constant-width vent or through a simple fan-resistance ventilation system.

CALLED BY: FVENT UVENT

FPLUME *specific* - Driver routine to calculate mass, enthalpy and product flows due to all plumes in the scenario being modeled. this routine is an interface between the catalog physics routines and ccfm.vent.

CALLS: GETSIZ INTERP MSGPRT PLUGO
COMMONS: CONST FIRE FLAGS MEMRY MEMRY2 PRODS ROOM SIZES VARMAP
CALLED BY: FBUILD

FROOM *specific* - This routine computes the rhs of the CCFM.VENTS ODE"S for the room, jroom. This routine is called by FBUILD once for each room in the simulation.

COMMONS: CONST FLAGS IO ROOM SIZES VARMAP
CALLED BY: FBUILD

FVENT *specific* - Driver routine to calculate mass, enthalpy and product flows due to all forced vents in the scenario being modeled. this routine is an interface between the catalog physics routines and CCFM.VENTS.

CALLS: FLOGO2 GETSIZ GETVAR VENTF
COMMONS: CONST FLAGS MEMRY MEMRY2 PRODS ROOM SIZES VENTS VNTSLB
CALLED BY: FBUILD

GETVAR *specific* - Routine to interface between CCFM.VENTS global data structures and vent (both natural and forced) data structures.

COMMONS: CONST SIZES VARMAP
CALLED BY: FVENT UVENT

INIT *specific* - Initialization routine.

CALLS: ERINIT GETPTR

COMMONS: CHECK COMTYP CONST FIRE INITS MFLAGS OUTS PRODS ROOM SIZES TITLS TOKEN1 TOKEN2 VENTS
CALLED BY: MAIN

INITMD *specific* - Initialize values required by the Model module.

CALLS: CPTIME NUMINF
LIB: MAX
COMMONS: CHECK CONST FIRE INITS IO MEMRY MEMRY2 MODCOM OUTS PRODS ROOM SIZES VENTS
CALLED BY: MAIN MENU1

MAIN *specific* - Main routine of the CCFM package.

CALLS: CPTIME INIT INITMD INITMM INPUT MODEL OUTWRP
COMMONS: COMTYP IO MFLAGS

MODEL *specific* - Driver routine for MODEL package.

CALLS: CPTIME DUMP FBUILD JACB MSGPRT OPENFL OUTPUT SETDE STATE1
LIB: DEBDF MIN
COMMONS: CONST FILES FIO FIRE FLAGS IO MIO MODCOM ROOM SIZES TITLS
CALLED BY: MAIN

MRES *generic* - This routine computes the mass flow in a fan system for a given resitance and pressure.

LIB: ABS SQRT
CALLED BY: FANFOR FANRES FANREV

PLUGO *generic* - Routine to calculate the mass, enthalpy and product flows due to one particular plume.

CALLED BY: FPLUME

RMVDUP *specific* - This routine removes duplicate entries from the sorted data quad-ruples (x(i),y(i),z(i),w(i) i=1 ... n). The parameter n is then adjusted to the new number of entries. Two entries i and i+1 are duplicates if x(i)=x(i+1).

CALLED BY: COMWL1 VENTHP

SETDE *specific* - This routine copies information stored in the solver data structure, y, into CCFM.VENTS global variables contained in the common block /VARMAP/ if the ODE formulation is changed, then only this routine needs to be modified to reflect the new structure of y.

COMMONS: CHECK CONST PRODS ROOM SIZES VARMAP
CALLED BY: FBUILD MENU1 MODEL

STATE1 *specific* - Print out the state of the simulation.

CALLS: GETSIZ NUMINF
COMMONS: CONST ERCOM ERRCHR FILES FIO FIRE FLAGS INITS IO MEMRY
COMMONS: MEMRY2 MFLAGS MIO MODCOM OUTS PRODS ROOM SIZES TITLS TOKEN1 TOKEN2 VENTS
CALLED BY: MENU1 MODEL

UVENT *specific* - Driver routine to calculate mass, enthalpy and product flows due to all unforced vents in the scenario being modeled. This routine is an interface between the catalog physics routines and CCFM.VENT.

CALLS: COMWL1 FLOGO2 GETVAR VENTHP
COMMONS: CONST FLAGS PRODS ROOM SIZES VARMAP VENTS VNTSLB
CALLED BY: FBUILD

VENTF *generic* - This routine calculates the flows (mass, enthalpy and product of combustion) through a fan duct system.

CALLS: FANRES MSGPRT
LIB: ABS SQRT
CALLED BY: FVENT

VENTHP *generic* - Calculation of the flow of mass, enthalpy, oxygen and other products of combustion through a vertical, constant-width vent in a wall segment common to two rooms. The subroutine uses input data describing the two-laye environment in each of the two rooms and other input data calculated in subroutine COMWL1.

CALLS: DELP HSORT MSGPRT RMVDUP
LIB: ABS EXP MAX MIN SQRT
CALLED BY: UVENT
CHECK : DUMP GENCHK INIT INITMD MENU1 NUMINF SETDE

A.3 OUTPUT Module Routines

OUTFLW *specific* - Print out a report containing various flows due to vents, plumes, etc.

CALLS: OUTWRN
COMMONS: FLAGS FLOWS IO MODCOM OUTS SIZES TITLS
CALLED BY: OUTPUT

OUTGEN *specific* - Print out a report containing relative pressure, layer height, temperatures and oxygen if modeled for each room.

CALLS: OUTWRN
COMMONS: CONST IO MFLAGS MODCOM OUTS SIZES TITLS VARMAP
CALLED BY: OUTPUT

OUTNUM *specific* - Print out a report containing numeric information such as number of iterations and CPU required for each time step.

CALLS: OUTWRN
COMMONS: IO MODCOM OUTS SIZES TITLS
CALLED BY: OUTPUT

OUTPLT *specific* - Print out a table of numbers used by an external plotting routine.

COMMONS: CONST FLOWS IO MODCOM SIZES VARMAP
CALLED BY: OUTPUT

OUTPRD *specific* - Print out a report containing all products being simulated for each time step

CALLS: OUTWRN
COMMONS: FLAGS IO MODCOM OUTS SIZES TITLS VARMAP
CALLED BY: OUTPUT

OUTPUT *specific* - Driver routine that determines whether printing should occur at the current time.

CALLS: EIGF FBUILD OUTFLW OUTGEN OUTNUM OUTPLT OUTPRD OUTUSR
COMMONS: FIO FLAGS MFLAGS MIO MODCOM OUTS SIZES
CALLED BY: MENU1 MODEL

OUTUSR *specific* - A sample routine that may customized by the user to fit her/his own needs.

COMMONS: IO MODCOM SIZES
CALLED BY: OUTPUT

OUTWRP *generic* The reports written by OUTFLW, OUTGEN, OUTNUM, OUTPRD and OUTDEB are written to one file. The routine OUTWRP sorts the information in this file by report type and copies the result to another file.

CALLS: LENGTH
LIB: CHAR ICHAR MAX
CALLED BY: MAIN MENU1

A.4 CHARACTER Manipulation/Conversion Routines

ALPHA *generic* - This logical function returns .true. if the input character, c is an alphabetic character (between 'A' and 'Z' or 'a' and 'z') and .false. otherwise.

CALLS: UPPER
CALLED BY: LEGAL

INTVAL generic - This routine converts a character string to an integer value

CALLS: LENGTH NUM
LIB: I1MACH ICHAR
CALLED BY: ARAYIN IBNCHK RDMENU RVAL

LENGTH *generic* - This function returns the position of the last non-blank character in a character string. This is different than the FORTRAN function, LEN. LEN returns the length of the space allocated to a character string. So if the following two statements appear in a FORTRAN program

```
        CHARACTER*80 CARD
        CARD='ABC'
```

then len(card) = 80 while length(card) = 3.

LIB: LEN
CALLED BY: DUMP INTVAL LEGAL LVAL MENU1 MSGPRT OPENFL OUTWRP RVAL UPPER

LVAL *generic* - This routine converts character string to a logical value.

CALLS: LENGTH UPPER
LIB: MIN
CALLED BY: LBNCHK

NOBLNK *generic* - This integer function returns the column of a character string containing the first nonblank character and a zero if the character string is completely blank.

LIB: LEN

NORMAL *generic* - This routine takes a mantissa of a floating point number defined and normalizes it so that the decimal is in the first column.

LIB: I1MACH INDEX LOG10 MIN
CALLED BY: RVAL

NUM *generic* - This is a logical function which returns true if a character string is a number between 0 and 9 and returns false otherwise.

LIB: ICHAR
CALLED BY: INTVAL LEGAL

UPPER *generic* - This routine converts any characters in a character string that are in lower case to upper case.

CALLS: LENGTH MSGPRT
LIB: CHAR ICHAR LEN
CALLED BY: ALPHA GETITM LVAL MENU1 RVAL

A.5 MEMORY Management Routines

CPYPTR *generic* - A routine in the memory manager package. Given a pointer to a block of memory, this routine copies that block to another portion of memory and returns a pointer to the new block.

CALLS: GETPTR
COMMONS: MEMRY MEMRY2
CALLED BY: ARAYIN RESIZ

DCODE *generic* - A routine in the memory manager package. Given a pointer to a block memory, this routine determines whether that pointer is valid.

CALLED BY: ARAYIN DELPTR RESIZ

DELPTR *generic* - A routine in the memory manager package. Given a pointer to a block of memory, this routine deletes that block.

CALLS: DCODE NCODE
COMMONS: MEMRY MEMRY2
CALLED BY: ARAYIN RESIZ

GETPTR *generic* - A routine in the memory manager package. This routine allocates a block of memory of a given size and type. The valid types are integer, floating point or character.

CALLS: NCODE
LIB: MOD
COMMONS: MEMRY MEMRY2
CALLED BY: ARAYIN CPYPTR INIT RESIZ

GETSIZ *generic* - This routine determines the size of a block of memory. This routine requires a pointer to the block and its type (integer, real or double precision). The initialization routine INITMM specifies how many integers correspond to a unit of both single and double precision memory.

COMMONS: MEMRY MEMRY2

CALLED BY: ARAYIN ARYTOK DUMP FPLUME FVENT GETVCT STATE1

INITMM *generic* - The initialization routine of the memory manager package. This routine initializes the memory manager by defining a first and last block which are used internally by the memory manager and cannot be deleted

CALLS: NCODE
COMMONS: MEMRY MEMRY2
CALLED BY: MAIN

NCODE *generic* - A routine in the memory manager package. It encodes the information portion of a block of memory so that the block can be checked later for validity i.e. when deleting a block of memory we make sure that the block is valid. the descriptor or information portion of a memory block contains pointers to the next and previous block, the block size and the block type. A fifth memory location contains an encoded piece of data defined by this routine. This is done as an error detection strategy to ensure that pointers to memory blocks are valid. See the memory manager section of this manual for a description of how memory blocks are organized.

CALLED BY: DELPTR GETPTR INITMM RESIZ

RESIZ *generic* - Resize a block of memory. The subroutine RESIZ resizes a block of memory. If there is not enough room where the block is located it will allocatr another block and copying the old block to the newly allocated block.

CALLS: CPYPTR DCODE DELPTR GETPTR NCODE
COMMONS: MEMRY MEMRY2
CALLED BY: ARAYIN ARYTOK

A.6 MISCELLANEOUS Routines

BRACK *specific* - This routine brackets the root of a function, so that the root lies in the interval [xa,xb]. The root contained in this interval is then refined using the routine RTSAF2. This subroutine is used in the forced vent calculation procedure.

CPTIME *specific* - Routine to calculate amount of computer time (cputim) in seconds used so far in a run. This routine will generally be different for each computer.

LIB: SECOND
CALLED BY: FBUILD INITMD MAIN MODEL

EIGF *generic* - Calculates the jacobian and its eigenvalues of the ODE being modeled. This routine is not required to simulate a fire scenario. It is used to examine the numerical properties of the problem being solved.

CALLS: FBUILD MAXMIN
LIB: F02AFF
COMMONS: FLAGS IO SIZES
CALLED BY: OUTPUT

ERINIT *generic* - Initializes the error reporting module.

COMMONS: ERCOM ERRCHR
CALLED BY: INIT

HSORT *generic* - This routine sorts the array, ra, in ascending order using a heap sort. Corresponding changes are made in the associated arrays rb, rc and rd.

CALLED BY: COMWL1 VENTHP

INDEXC *generic* - Constructs an index array, i(j), so that the array of character strings a(i(j)) is in ascending order for j=1, ..., n.

INDEXI *generic* - Constructs an index array, i(j), so that the array of integers a(i(j)) is in ascending order for j=1, ..., n.

INTERP generic - This routine interpolates a table of numbers found in the arrays, x and y.

CALLED BY: FANFOR FANRES FANREV FPLUME

INTV *generic* For a specified real number this routine determines the interval of an array (in ascending order) where the number occurs.

LEGAL *generic* - this logical function returns .true. if file is a legal file name false otherwise.

CALLS: ALPHA LENGTH MSGPRT NUM
COMMONS: COMTYP
CALLED BY: FBNCHK OPENFL

MSGPRT *generic* - Prints out a message and the messages error level: informative, warning, error or fatal error.

CALLS: LENGTH
LIB: IABS MIN
COMMONS: ERCOM ERRCHR
CALLED BY: ARAYIN ARFRSH CRDMRG FBNCHK FPLUME GETHLP IBNCHK INPUT LBNCHK LEGAL MENU0 MENU1 MODEL OPENFL PRTMEN RBNCHK RDMENU RFRSH RMVBLK RTSAF2 SCRNIN UPPER VENTF VENTHP

NUMINF *specific* - This routine determines the number of inside rooms, outside rooms, products, vents and fires.

LIB: MAX
COMMONS: CHECK SIZES
CALLED BY: GENCHK INITMD MENU1 STATE1

OPENFL *generic* - This routine opens a file for input or output depending on the input parameter values.

CALLS: LEGAL LENGTH MSGPRT RMVBLK
LIB: LEN MIN
CALLED BY: MENU1 MODEL

RTSAF2 *specific* - This routine finds a zero of a function using a modification of newton's method.

CALLS: MSGPRT
LIB: ABS FUNCD
CALLED BY: FANRES

B. CCFM Global Data Structures Definitions

This appendix contains a listing of CCFM.VENTS global data structures or equivalently THE CCFM.VENTS common blocks. Each common block entry contains a listing of FORTRAN declarations and a short description of each variable in the common block. Each common block description also contains a listing of subroutines that reference the common block. This information is useful to the person who wishes to modify a common block and wants to know which routines access it.

CHECK

A variable in the common block CHECK is set whenever a room, vent, fire or product are defined or referenced by a CCFM.VENTS command. The **ROOM**, **VENT**, **FIRE** and **INITP** commands define rooms, vents, fires and products respectively. The **VENT** and **FVENT** commands also specify two rooms that are to be connected by a vent or fan-duct system. The purpose of the CHECK common block then is to provide information to the generic error checking subroutine GENCHK so that the user may be warned if a room was not defined but was referenced by another CCFM.VENTS command.

```
      INTEGER CHKFIR(MXFIRE), CHKRM(-MXORM:MXIRM), CHKVNT(MXVNTS)
      INTEGER CHKPRD(MXPRD)
      COMMON /CHECK/CHKFIR, CHKRM, CHKVNT, CHKPRD
      SAVE /CHECK/
```

Each variable in CHECK is defined in the same way. A value of 0 for a CHECK variable means that the item was never defined, a value of 1 means that the item is defined while a value of -1 means that the item was referenced by a CCFM.VENTS command but is not defined.

CHKFIR	Definition status for fires
CHKRM	Definition status for rooms
CHKVNT	Definition status for vents
CHKPRD	Definition status for products of combustion

COMTYP

This common block indicates what computer is running CCFM.VENTS. Various possibilities are defined in parameter statements. Most routines in CCFM.VENTS do not require this information. Any routine that does depend on the type of computer that is executing CCFM.VENTS would use this common block. One example of where this common block is needed is the main routine since it opens the terminal screen for both input and output. In general the exact method for opening "terminal" input and output files will vary from one computer to the next.

```
      INTEGER MAC, IBMPC, NOS, NOSVE, CYB205, COMPAQ, OTHER, CON
      PARAMETER (MAC=1,IBMPC=2,NOS=3,NOSVE=4,CYB205=5,COMPAQ=6,CON=7)
      PARAMETER (OTHER=0)
      INTEGER COMPTR
      COMMON /COMTYP/COMPTR
      SAVE /COMTYP/
```

COMPTR	Type of computer that is running CCFM.VENTS

USED BY: GETHLP INIT LEGAL MAIN

CONST

The common block /CONST/ contains the values of various constants.

```
      DIMENSION ATOL(5), RTOL(5)
      COMMON /CONST/ ALAMR, ALAMT, ALAMW,
     1               R, CV, CP, G, GAM,
     2               PDATM, DDATM, TDATM,
     3               ATOL, RTOL
      SAVE /CONST/
```

ALAMR Fraction of the total energy release rate of the fire radiated by the combustion zone and plume (default 0.35).

ALAMT Fraction of the total energy release rate of the fire lost to the bounding surfaces of the room of fire origin by all modes of heat transfer (default 0.8).

ALAMW Define the "enthalpy of buoyancy" of a uniform temperature vent flow as the enthalpy of the flow computed relative to the temperature in the receiving room local to flow penetration. Then lamw is the fraction of the enthalpy of buoyancy lost to the bounding surfaces of the receiving room (default 0.6).

R ideal gas law constant

CV specific heat at constant volume

CP Specific heat of air at constant pressure (default 1000.). [W•s/(kg•K)]

G acceleration of gravity

GAM The ratio of CV and CP; 1.4

PDATM datum pressure, the absolute hydrostatic pressure at the datum elevation (default 101325.). [p] = [N/m^2] = [kg/(m•s)^2]

DDATM datum density (default 1.2). [kg/m^3]

TDATM datum temperature - not currently used in this version of CCFM.VENTS

ATOL absolute error tolerances - used by ODE solver

RTOL relative error tolerances - used by ODE solver

USED BY: DUMP FPLUME FROOM FVENT GETVAR INIT INITMD MENU1
MODEL OUTGEN OUTPLT SETDE STATE1 UVENT

ERCOM - ERRCHR

The common blocks /ERCOM/ and /ERRCHR/ are used by the printing subroutines MSGPRT

```
C
      CHARACTER*20 CLEVEL(4)
      COMMON /ERCOM/ IOERR1, IOERR2, IOLEVL, IOLVSV
      COMMON /ERRCHR/ CLEVEL
      SAVE /ERCOM/
      SAVE /ERRCHR/
```

IOERR1 A FORTRAN i/o unit number used to print out messages by the subroutine ERPRt

IOERR2 Not used at present.

IOLEVL messages with a level greater than or equal to IOLEVL will be printed. example When loading a file with the **LOAD** command IOLEVL is set to 1 so routine confirmation messages will not be printed but warnings, errors and fatal errors will be. The user may set this value with the **LEVEL** command.

IOLVSV When the LOAD command is executed a copy of IOLEVL into this variable. Then the value of IOLEVL is increased to 1. This keeps routine messages from being printed out while loading an input data file. After the load is completed IOLVSV is copied back into IOLEVL.

CLEVEL The character string array containing the name of the message levels, e.g. message, warning, error, fatal error.

USED BY: DUMP ERINIT INPUT MENU0 MENU1 MSGPRT STATE1

FILES

This common block contains the name of various files used by CCFM.

```
CHARACTER*20   PRINTF, PRINTG, PLOTF, DUMPF, LOADF
CHARACTER*64   PATH
CHARACTER*84   FTEMP
COMMON /FILES/ PATH, PRINTF, PRINTG, PLOTF, DUMPF, LOADF
SAVE /FILES/
```

PATH This is not a file name but a path name that is used whenever a file is opened fo rinput or output.

PRINTF Scratch output file

PRINTG Final output file

PLOTF Plot file

DUMPF Dump file

LOADF Load file

USED BY: DUMP MENU1 MODEL STATE1

FIRE

This common block contains information describing a fire. Entries in the common block define a height, room location and pointers to arrays that define the size of the fire at various times.

```
INTEGER VRMFO, VQFIRE
COMMON /FIRE/ VHFIRE(MXFIRE), VRMFO(MXFIRE), VQFIRE(MXFIRE,2)
SAVE /FIRE/
```

VHFIRE(i) The height of the i'th fire above the floor

VRMFO(i) The room that the i'th fire is in, valid entries are 1 to MXFIRE.

VQFIRE VQFIRE(i,1) is a pointer to a time array for the i'th fire, VQFIRE(i,2) is a pointer to a an array that contains the energy release rates of the i'th fire. The units for fire size are watts.

USED BY: DUMP FPLUME INIT INITMD MENU1 MODEL STATE1

FLAGS

Various flags used by CCFM. Note: yes = 1, no = 0.

```
COMMON /FLAGS/IPRTAL,IPLOT,IDISC,IEIG,IFLOGO
SAVE /FLAGS/
```

IPRTAL Printing to occur at every internal solver step

IPLOT Plotting flag

IDISC If set to 1 then a heuristic (coded in the routine DISC) will be used to predict and step over non-analytic solution behavior

IEIG Flag to determine whether eigenvalues of jacobean of rhs of ode are to be calculated. (note: this option is not implemented at the present time)

IFLOGO not a flag, if IFLOGO = i then use the i'th algorithm for determining where flow's go. The only valid value in the present version of CCFM.VENTS is IFLOGO=2. This flag can be used whenever any upgrade to the FLOGO algorithm is implemented.

USED BY: DUMP EIGF FBUILD FPLUME FROOM FVENT MENU1 MODEL OUTFLW OUTPRD OUTPUT PLUGO STATE1 UVENT

FLOWS

This common block records flow due to forced vents, un-forced vents, fire plumes and the total for both the upper and lower layers in each room . The type of flow recorded is mass, enthalpy, and products of combustion.

```
DIMENSION FLWF(MXTRM,MXPRD+2,2), FLWU(MXTRM,MXPRD+2,2)
DIMENSION FLWP(MXTRM,MXPRD+2,2), FLWTOT(MXTRM,MXPRD+2,2)
COMMON /FLOWS/FLWF,FLWU,FLWP,FLWTOT
SAVE /FLOWS/
```

FLWF Flows due to forced vents

FLWU Flows due to unforced vents

FLWP Flows due to plumes

FLWTOT Total flows

USED BY: FBUILD OUTFLW OUTPLT

FLWPTRS

This is not a common block but a collection of parameters used to access the flow arrays used in the common block, /FLOWS/.

```
INTEGER L,U,M,Q,O,P
PARAMETER (L=1,U=2,M=1,Q=2,O=3,P=3)
```

L Lower Layer

U	Upper Layer
M	Mass
Q	Enthalpy
O	Oxygen
P	Products of Combustion

INITS

A collection of arrays used to record the initial conditions of solution variables.

```
DIMENSION PFLOR0(MXIRM), YLAY0(MXIRM)
DIMENSION TEMP0(MXIRM), RHOL0(MXIRM), RHOU0(MXIRM)
COMMON /INITS/ PFLOR0, YLAY0, RHOL0, RHOU0, TEMP0
SAVE /INITS/
```

PFLOR0	initial pressure in pascals
YLAY0	initial layer height
RHOL0	initial lower layer density
RHOU0	initial upper layer density

USED BY: FBUILD OUTFLW OUTPLT

IO

FORTRAN I/O unit numbers used by CCFM.

```
COMMON /IO/ IN1, IN2, IOUT1, IOUT2, IOUT3, IOUT4, IOUTER
SAVE /IO/
```

IN1	unit number for input from the keyboard
IN2	unit number for input from the menu definition file (MENU989)
IOUT1	unit number for output to the terminal screen
IOUT2	unit number for output to the scratch file
IOUT3	unit number for output to the plot data file
IOUT4	unit number for final output file
IOUTER	unit number for output to the error file - not used

USED BY: EIGF FBUILD FROOM INITMD MAIN MENU1 MODEL OUTFLW OUTGEN OUTNUM OUTPLT OUTPRD OUTUSR OUTWRN RFRSH STATE1

MEMPRM

Parameters used internally by the memory manager. Each block of memory allocated by GETPTR has an over head of BLKSIZ integers. This extra space is used to record information such as the location of the next block, location of the previous block, size of this block and type of block (integer, floating point) and a checksum. The parameters defined below provide a convenient way to access this information. For example if IXPTR is a pointer to an integer array allocated by GETPTR then MEMORY(IXPTR-BLKSIZ + NXT) is a pointer to the next block of memory, MEMORY(IXPTR-BLKSIZ+SIZ) is the size of this block. The parameters defined here are used internally by the memory manager.

```
INTEGER NXT, PRV, SIZ, TYP, COD, BLKSIZ, PAD
PARAMETER (NXT=0,PRV=1,SIZ=2,TYP=3,COD=4,PAD=5,BLKSIZ=6)
```

NXT	offset that contains location of pointer to next memory block
PRV	offset that contains location of pointer previous memory block
SIZ	offset to location that contains of size of block
TYP	offset that contains location of type of block
COD	offset to locatoin of en information for memory block
PAD	not used
BLKSIZ	size of over head block

MEMRY

All memory allocations are made from this common block. Any routine that uses a pointer to a block of memory defined by GETPTR must include this common block within it. FLTINT has a value of 1 (2) for single (double) precision versions of CCFM.VENTS. The present version of the memory manager used by CCFM.VENTS does not support allocation of character arrays. The hooks are included however for a future implementation.

```
      INTEGER FLTINT
C
C*** FLTINT=1 FOR SINGLE PRECISION
C    FLTINT=2 FOR DOUBLE PRECISION
C
      PARAMETER (FLTINT=2)
C     PARAMETER (FLTINT=1)
      PARAMETER (MXI=2000,MXF=MXI/FLTINT,MXC=1)
      DIMENSION MEMORY(MXI), FLTMEM(MXF)
      CHARACTER*1 CHRMEM(MXC)
      EQUIVALENCE (MEMORY,FLTMEM)
      COMMON /MEMRY/ MEMORY
      COMMON /MEMRY2/ CHRMEM
      SAVE /MEMRY/
      SAVE /MEMRY2/
```

MEMORY integer array used for integer memory allocation
FLTMEM floating point array (equivalenced to MEMORY) used for floating point memory allocation
CHRMEM character array used for character memory allocation - not used in present version of CCFM.VENTS

USED BY: ARAYIN ARYTOK CPYPTR DELPTR DUMP FPLUME FVENT GETPTR GETSIZ GETVCT INITMD INITMM MENU1 RESIZ STATE1

MENSTF

The variables in MENSTF are used to process CCFM.VENTS commands. The are used in the following way:

1. The command line is read into the character variable CARD.
2. CARD is then parsed by subroutine PARSE, i.e. the beginning and ending of each set of contiguous non-blank characters is recorded in the integer arrays, SB and SE (string begin and string end). The number of tokens found is saved in NTOK.
3. The first token found on CARD is passed to the subroutine GETITM to determine what command (if any) was entered. If the command was legal then GETITM returns the commands i.d. number otherwise an appropriate error message is printed and the user is given another chance.
4. If the command number found by GETITM in step 3. is negative then the command was generic and control is passed to MENU0. If the command number is positive then it is a CCFM command and control is passed to MENU1.
5. Now proceed with step 1. unless MENU1 signals that processing should be terminated. MENU1 signals that the CCFM simulation should begin by returning a value of NTOK=0.

```
      CHARACTER*120 CARD
      CHARACTER*10 PROMPT
      INTEGER IN,OUT,LE,LT,GE,GT,NIL
      PARAMETER (IN=1,OUT=2,LE=1,LT=2,GE=1,GT=2,NIL=0)
      PARAMETER (MXTOK=40)
      INTEGER SB(MXTOK), SE(MXTOK)
C
```

```
      COMMON /TOKEN1/SB, SE, NTOK, LPRMPT, IOVER
      COMMON /TOKEN2/CARD,PROMPT
      SAVE /TOKEN1/
      SAVE /TOKEN2/
```

SB(i) column number in CARD of beginning of i'th token. If SB(i)=0 then the i'th token is not present.

SE(i) column number in CARD of end of i'th token. If SB(i)=0 then SE(i) is un-defined.

NTOK number of tokens

CARD character variable containing command line that was entered.

USED BY: ARAYIN ARYTOK FBNCHK IBNCHK INIT INPUT LBNCHK MENU0 MENU1 RBNCHK RDMENU SCRNIN STATE1

MENUS

This common block is used by the INPUT package of CCFM.VENTS to define menus. Each time a CCFM.VENT command is entered the following events occur:

```
      INTEGER MENMAX, COLMAX, ROWMAX, SIZSTK, STKPTR
      PARAMETER (ITMMAX=50,COLMAX=7,MENMAX=1,ROWMAX=10,SIZSTK=1)
C
      INTEGER NBEG(MENMAX,COLMAX),NCOL(MENMAX),MENSTK(SIZSTK)
      INTEGER DESPTR(ITMMAX,2),HLPPTR(ITMMAX,2)
      COMMON /MENUS/NBEG,NCOL,NMENU,IMENU,STKPTR,MENSTK,HLPPTR,DESPTR
C
      CHARACTER*40 DESPAR(ITMMAX*4)
      CHARACTER*20 MENLST(ITMMAX,2),TITLE(MENMAX),MENLS2(ITMMAX)
      COMMON /MENU2/MENLST,MENLS2,TITLE,DESPAR
      SAVE /MENUS/
      SAVE /MENU2/
C
      INTEGER SRTLST(ITMMAX),ROWNUM(ITMMAX),COLNUM(ITMMAX),N2CNT
      INTEGER COMPRG(ITMMAX),COMLST(ITMMAX)
      COMMON /MENU4/SRTLST, COMLST, ROWNUM, COLNUM, N2CNT, COMPRG
      SAVE /MENU4/
```

MENLST	LINEAR LIST OF MENU CHOICES
TITLE(I)	TITLE OF I'TH MENU
NBEG(I,J)	INDEX OF FIRST ITEM IN I'TH MENU IN IN J'TH COLUMN
NCOL(I)	NUMBER OF COLUMNS IN I'TH MENU
NMENU	NUMBER OF MENUS
IMENU	MENU NOW BEING PROCESSED
MNSTRT(I)	INDEX OF FIRST ITEM OF MENU I IN MENLST

Routines that affect data structures found in this common block are: PRTMEN, GETITM, RDMENU

MFLAGS

Generic flags used by the input package. If flag is set then variable is 1 otherwise the variable is 0.

```
 COMMON /MFLAGS/ IBRIEF, IBRFSV, ITRACE, ITTY, IBATCH,
1                IEOF, ILOAD, TLOAD
 SAVE /MFLAGS/
```

IBRIEF	Are menus printed out after an entered command
IBRF2	not used
ITRACE	Trace option - not used
ITTY	are results displayed to the terminal screen
IBATCH	This flag indicates whether the program is running in the foreground or in the background. The main difference in how CCFM.VENTS runs in these two modes is how it treats an end of file. If IBATCH is off then it is assumed that the program is being run interactively. So when an end of file occurs the input file is rewound and more input is solicited. If an end of file is encountered while in IBATCH is on CCFM.VENTS starts executing the fire scenario as if the command BEGIN had been entered.

USED BY: ARAYIN GETHLP GETITM INPUT PRTMEN RDMENU RFRSH SCRNIN

MIO

Fortran I/O unit number used by the input package

```
 CHARACTER*32 FMENU
 COMMON /MIO/MIN1,MMENU,MHELP,MSAVE,MLOAD,MDUMP
 COMMON /FIO/FMENU
 SAVE /MIO/
 SAVE /FIO/
```

MIN1	unit number used for inputting data
MMENU	unit number for reading in menu definition file (MENU989)
MHELP	unit number used to access help information
MSAVE	storage used to save copy of input unit number during LOAD command
MLOAD	unit number used load data files into memory
MDUMP	unit number used to dump CCFM.VENTS fire scenarios to data files

USED BY: ARAYIN GETHLP INPUT MENU0 MENU1 MODEL OUTPUT RDMENU RFRSH SCRNIN STATE1

MODCOM

Common block used to store information related to the solution of CCFM's ODE's.

```
 DIMENSION Y(MAXDE)
 DIMENSION YPRIME(MAXDE)
 INTEGER PRTTIM, DMPTIM
 COMMON /MODCOM/ TSTART, TCUR, TSTOP, TOUT,
1                Y, YPRIME,
2                DPRINT, DDUMP, TPRINT, TDUMP,
3                CPST, CPCUM, CPOVST, CPOVCM, CPP, CPOVP,
5                PRTTIM, DMPTIM,
4                TBEF, TAFT, TVNTCM, TVNTST
 SAVE /MODCOM/
```

TSTART	starting simulation time in seconds
TCUR	current time in seconds
TSTOP	stopping time in seconds
TOUT	next time in seconds for which printed output will occur
Y	solution of ODE
YPRIME	derivative of solution of ODE, calculated by the subroutine FBUILD
DPRINT	print interval time in seconds
DDUMP	dump interval time in seconds
CPST	CPU time for the last printing step
CPCUM	CPU time since the beginning of the simulation (cumulative time)
CPOVST	Overhead CPU time during the last print interval. Overhead is the work performed in printing solutions.
CPOVCM	Cumulative overhead CPU time.
CPP	time for last printing interval
CPOVP	overhead time for last printing step
PRTTIM	time interval between printed output in seconds
DMPTIM	time interval between dump output in seconds
TBEF	time since beginning of run before call to solver
TAFT	time since beginning of run before call to solver
TVNTCM	cumulative time in vent routines
TVNTST	time in vent routines for last print interval

USED BY: DUMP FBUILD INITMD MENU1 MODEL OUTFLW OUTGEN OUTNUM OUTPLT OUTPRD OUTPUT OUTUSR STATE1

OUTS

Contains flags to determine which reports to print out, information on how hard the solver is working.

```
INTEGER RPTS(MXRPT), ILINE(MXRPT), PAGELN
PARAMETER (PAGELN=60)
COMMON /OUTS/ INSTEP, IPSTEP, RPTS, ILINE
SAVE /OUTS/
```

INSTEP	number of internal steps.
IPSTEP	number of printing steps
RPTS	an array of indicator values, if RPTS(i)=1 then the i'th report will be printed
ILINE	the line number of the report that was last printed, used for pagination purposes

USED BY: DUMP INIT INITMD MENU1 OUTFLW OUTGEN OUTNUM OUTPRD OUTPUT STATE1

PRODS

Contains initial product concentrations for each room, layer and product.

```
      DIMENSION ACON(MXORM,MXPRD)
      DIMENSION PFACTS(MXPRD), PRDS0(MXIRM,MXPRD,2)
      COMMON /PRODS/ PFACTS, PRDS0, ACON
      SAVE /PRODS/
```

PFACTS	For the i'th product, PFACTS(i)*Q determines how much of the i'th product will be given off in a fire of size Q. In the case of oxygen this term gives the amount absorbed by the fire.
PRDS0(i,j,k)	Initial j'th product concentration in the i'th room in the both the upper (k=2) and lower layers (k=1).
ACON(i,j)	Initial j'th product concentration in the i'th outside room.

USED BY: DUMP FPLUME FVENT INIT INITMD MENU1 SETDE STATE1 UVENT

ROOM

Contains a description of room properties.

```
      COMMON /ROOM/ VAROOM(MXIRM), VHCEIL(MXIRM), VYFLOR(-MXORM:MXIRM),
     2              VYCEIL(MXIRM), ADEN(MXORM), APRESS(MXORM)
      SAVE /ROOM/
```

VAROOM(i)	area of the i'th room
VHCEIL(i)	height of ceiling above floor in i'th room
VYFLOR(i)	height of floor above datum height in i'th room
VYCEIL(i)	height of ceiling above datum height in i'th room, note: VYCEIL(i) = VYFLOR(i) + VHCEIL(i).
ADEN(i)	density of i'th outside environment
APRESS(i)	pressure above datum pressure of i'th outside environment

USED BY: DUMP FPLUME FROOM FVENT INIT INITMD MENU1 MODEL SETDE STATE1 UVENT

SIZES

Contains the maximum sizes allowed and the actual number for quantities such as number of fires, number of rooms, number of products, etc.

```
      PARAMETER (MXIRM=9,MXORM=4,MXTRM=MXIRM+MXORM)
      PARAMETER (MXPRD=5,MXRMDE=4+2*MXPRD,MAXDE=MXIRM*MXRMDE)
      PARAMETER (MXQ=100,MXFIRE=1)
      PARAMETER (MXDUMP=10,MXVNTS=20)
      PARAMETER (MXELEV=10,MXSLAB=MXELEV-1,MXFLOW=2)
      PARAMETER (MXRPT=6)
      PARAMETER (MXDISC=4*MXIRM+4*MXVNTS+100)
      INTEGER   YES,NO
      PARAMETER (YES=1,NO=0)
      COMMON /SIZES/NRMDE, NDES, NORM, NIRM, NTRM, NPROD, NFIRE,
     1              NVENTS,NORM0,NIRM0,NPROD0,NFIRE0,NVNTS0
      SAVE /SIZES/
```

MXIRM, [NIRM]	Maximum [actual] number of inside rooms
MXORM, [NORM]	Maximum [actual] number of outside environments
MXPRD, [NPROD]	Maximum [actual] number of products of combustion
MXTRM, [NTRM]	Maximum [actual] room total, the sum of inside and outside rooms
MXRMDE, [NRMDE]	Maximum [actual] number of ODE's per room, 4 plus 2 times the number of products
MAXDE, [NDES]	Maximum [actual] number of ODE's being solved, the maximum number of inside rooms times the maximum number of ODE's per room
MXQ	obsolete, no longer used
MXFIRE, [NFIRE]	maximum [actual] number of fires
MXDUMP	
MXVNTS, [NVENTS]	maximum [acatual] number of vents
MXELEV	maximum number of elevations, need be only seven, (top and bottom of vent, two layers and three neutral planes)
MXSLAB	maximum number of slabs, one less than the number of elevations
MXRPT	maximum number of output reports
MXDISC	maximum number of discontinuities

USED BY: DUMP EIGF FBUILD FPLUME FROOM FVENT GENCHK GETVAR INIT INITMD MENU1 MODEL NUMINF OUTFLW OUTGEN OUTNUM OUTPLT OUTPRD OUTPUT OUTUSR SETDE STATE1 UVENT

TITLS

This common block contains titles that can optionally be printed out to Dump, Output and/or plot files.

```
      CHARACTER*80 TITLES
      CHARACTER*6 TTYPE
      DIMENSION TITLES(3), TTYPE(3)
      INTEGER DMPTTL, OUTTTL, PLTTTL
      PARAMETER (DMPTTL=1,OUTTTL=2,PLTTTL=3)
      COMMON /TITLS/TITLES,TTYPE
      SAVE /TITLS/
```

TITLES	text of titles
TTYPE	type of title

USED BY: DUMP INIT MENU1 MODEL OUTFLW OUTGEN OUTNUM OUTPRD STATE1

VARMAP

Current values of CCFM.VENTS variables. This common block contains values that describe the geometric and physical properties of a given fire scenario. This common block is filled in by the subroutine SETDE. Some of the same information such as layer heights and relative pressure are contained in arrays used by the ODE solver. All physical routines should obtain this information from these variables, since the ODE formulation may change.

```
      DIMENSION ZDP(-MXORM:MXIRM),      ZP(-MXORM:MXIRM)
      DIMENSION ZDRHO(-MXORM:MXIRM,2), ZRHO(-MXORM:MXIRM,2)
      DIMENSION ZDM(-MXORM:MXIRM,2),    ZM(-MXORM:MXIRM,2)
      DIMENSION ZDTM(-MXORM:MXIRM,2),  ZTM(-MXORM:MXIRM,2)
C
      DIMENSION ZVOL(-MXORM:MXIRM,2),  ZYLAY(-MXORM:MXIRM)
      DIMENSION ZHLAY(-MXORM:MXIRM)
      DIMENSION ZYFLOR(-MXORM:MXIRM),    ZYCEIL(-MXORM:MXIRM)
      DIMENSION ZAREA(-MXORM:MXIRM),     ZHCEIL(-MXORM:MXIRM)
C
      DIMENSION ZCON(-MXORM:MXIRM,2,MXPRD)
C
      COMMON /VARMAP/   ZDP,     ZP, ZYLAY, ZHLAY,
     1                 ZDRHO,   ZRHO,  ZDTM,   ZTM,
     2                   ZDM,     ZM,
     2                  ZVOL, ZYFLOR, ZHCEIL, ZYCEIL, ZAREA,
     3                  ZCON
      SAVE /VARMAP/
```

ZDP, [ZP]	Floor pressure relative to PDATUM [0]
ZDRHO, ZRHO	Layer density relative to DDATUM [0]
ZDTM, ZTM	Layer temperature relative to DDATUM [0]
ZM	Layer mass
ZDM	Layer mass relative to some datum mass. This value is not calculated in the version of CCFM.VENTS
ZVOL	Layer volume
ZAREA	Floor area
ZYFLOR	Height of floor above datum elevation
ZYLAY, [ZHLAY]	Height of layer above datum [floor] elevation
ZYCEIL, [ZHCEIL]	Height of ceiling above datum [floor] elevation

USED BY: DUMP FPLUME FROOM GETVAR OUTGEN OUTPLT OUTPRD SETDE UVENT

VNTSLB

Record information about flow between a pair of rooms through an un-forced vent.

```
      DIMENSION YVELEV(10), DPV1M2(10)
      INTEGER DIRS12(10),NVELEV
      COMMON /VNTSLB/YVELEV,DPV1M2,DIRS12,NVELEV,IOUTF
      SAVE /VNTSLB/
```

YVELV	A list of heights sorted in increasing order of the elevations of interest, (top and bottom of vents, layer heights and neutral planes if any). Note: layer and neutral plane heights will appear in this list only if the occur between the top and bottom of the vent.
DPV1M2	An array of pressure differences, DPV1M(i) is the difference in pressure at the height specified by YVELV(i).
DIRS12	An integer array indicating the direction of flow.
NVELEV	The number of observations contained in each of the arrays: YVELV,

DPV1M2 and DIRS12.

IOUTF

USED BY: FVENT UVENT

VENTS

Contains information describing the vents in CCFM.

```
DIMENSION VNTSTF(MXVNTS,6), IVNTS(MXVNTS,6)
DIMENSION IVCASE(MXVNTS)
COMMON /VENTS/VNTSTF,IVNTS,IVCASE
SAVE /
```

VNTSTF(i,j) If vent i is an unforced vent then

VNTSTF(i,1) height of the bottom of the i'th vent above the floor with respect to the from room

VNTSTF(i,2) height of the top of the i'th vent above the floor with respect to the from room

VNTSTF(i,3) area of the vent

If vent i is a forced vent then

VNTSTF(i,1) height of the fan system above the floor in the from room

VNTSTF(i,2) height of the fan system above the floor in the to room

VNTSTF(i,3) resistance of the fan system

IVNTS(i,j) IVNTS(i,1) is the from room for the i'th vent

IVNTS(i,2) is the to room for the i'th vent

IVNTS(i,3) is the type of vent; IVNTS(i,3) = 1 then the vent is un-forced, IVNTS(i,3) = 2 then the vent is forced

IVNTS(i,4) for forced fan systems this variable contains a pointer to the pressure array defining the fan curve

IVNTS(i,5) for forced fan systems this variable contains a pointer to the volume flow array defining the fan curve.

USED BY: DUMP FVENT INIT INITMD MENU1 STATE1 UVENT

C. CCFM Menu Definition File

The Menu Definition File (MDF) for CCFM.VENTS is listed following this paragraph. It provides a means for defining the menus used by CCFM.VENTS. The MDF defines the spelling of each command, its position in the menu and any help or parameter description data to be printed out by CCFM.VENTS. This file is abbreviated by only including parameter descriptions help information for the command **CONS**. This information is the same as that given in section 5 of the User's Reference Guide.

```
;   THE INPUT MODULE READS IN THIS FILE AND USES IT TO CONSTRUCT
;   THE MENU FOR THE CONSOLIDATED MODEL.  THE MENU CONSISTS OF COMMANDS
;   ARRANGED IN ROWS AND COLUMNS.  EACH COMMAND IS A SEQUENCE OF ONE
;   OR MORE NON-BLANK CHARACTERS.  THE SPELLING OF A COMMAND MAY BE
;   CHANGED WITHOUT RE-COMPILING THE CONSOLIDATED MODEL.  IF TWO
;   COMMANDS ARE REARRANGED THEN A CORRESPONDING CHANGE MUST BE MADE
;   IN THE SUBROUTINE MENU1 IN THE INPUT MODULE.
;
;    SUMMARY OF MENU DESCRIPTION COMMANDS (MDC)
;
;   MDC                     DESCRIPTION

;   .NM TITLE        START A NEW MENU.  TITLE MAY BE A CHARACTER STRING
;                    UP TO 74 CHARACTERS LONG.  IMBEDDED BLANKS ARE ALLOWED.

;   .NM CTITLE       START A NEW COLUMN OF MENU COMMANDS.  CTITLE IS A
;                    CHARACTER STRING UP TO 10 CHARACTERS LONG.  IMBEDDED
;                    BLANKS ARE NOT ALLOWED

;   .C COM1 NUM      A MENU COMMAND.  COM1 IS A CHARACTER STRING  UP TO 10
;                    CHARACTERS LONG WITH NO IMBEDDED BLANKS.  NUM IS AN INTEGER
;                    INDICATING WHAT ELEMENT OF A COMPUTED GO TO STATEMENT WILL
;                      BE EXECUTED WHEN THE COMMAND COM1 IS TYPED

;    .PDB            BEGINNING OF PARAMETER DESCRIPTIONS
;    .PDE            END OF PARAMETER DESCRIPTIONS
;                    TEXT APPEARING BETWEEN .PDB AND .PDE WILL APPEAR IN CCFM.VENT'S
;                    SCREEN INPUT MODE

;   .HB              BEGINNING OF HELP TEXT
;   .HE              END OF HELP TEXT
;                    THE PURPOSE OF COMMAND HELP TEXT IS TO PROVIDE A
;                    MORE EXTENSIVE DESCRIPTION OF THE COMMAND GIVEN
;                    BY THE PREVIOUS .C COMMAND.
;
;   .DF              INDICATES THAT THE REST OF THE TEXT IN THIS FILE
;                    WILL CONSIST OF CCFM MENU COMMANDS DEFINING A
;                    DEFAULT CASE.

;     ;              A SEMI-COLEN ';' INDICATES A COMMENT.  TEXT AFTER
;                    A ';' IS IGNORED BY THE INPUT MODULE.


.NM CCFM.VENT               ; NEW MENU WITH ITS TITLE
                ; YOU CAN HAVE BLANK LINES FOR SPACING
```

```
.NC CONSTANTS

.C CONS 1
.PDB
FRACTION OF QFIRE RADIATED FROM FIRE AND PLUME
TOTAL FRACTION OF QFIRE TO SURFACES IN FIRE ROOM
FRACTION OF VENT-INFLOW ENTHALPY TO SURFACES
CP - SPECIFIC HEAT OF AIR [W*S/(KG*T)]
DATUM ABSOLUTE PRESSURE [PASCAL]
DATUM DENSITY [KG/M**3]
.PDE
.HB
CONS   lamr, lamt, lamw, cp, pdatm, ddatm

This command is used to specify various constants used by
CCFM.VENTS.

lamr   Fraction of the total energy release rate of the fire
       radiated by the combustion zone and plume (default
       0.35).
lamt   Fraction of the total energy release rate of the fire
       lost to the bounding surfaces of the room of fire
       origin by all modes of heat transfer (default 0.8).
lamw   Define the "enthalpy of buoyancy" of a uniform
       temperature vent flow as the enthalpy of the flow
       computed relative to the temperature in the receiving
       room local to flow penetration.  Then lamw is the
       fraction of the enthalpy of buoyancy lost to the
       bounding surfaces of the receiving room (default 0.6).
cp     Specific heat of air at constant pressure (default
       1000.).  [W•s/(kg•K)]
pdatm  datum pressure, the absolute hydrostatic pressure at
       the datum elevation (default 101325.).  [p] = [N/m2] =
       [kg/(m•s)2]
ddatm  datum density (default 1.2).  [kg/m3]


.HE

.C NUM 4
.C CHECK 12
.C CONV 14
.C PARMS 13
.C FACTOR 8
.C AMB 5
.PDB

.NC MODEL

.C ROOM 11
.C FIRE 7
.C VENT 10
.C FVENT 9
.C INITP 6
.C INIT 17
```

```
.C AUTO 18

.NC OUTPUT

.C PATH 2
.C FILES 22
.C TIME 15
.C REPORT 27
.C TITLE 25
.C VALUES 31

.NC MISC

.C OPTIONS -6
.C BEGIN 16
.C QUIT -5
.C LOAD 29
.C RERUN 28
.C DUMP 30
.C HELP -7

.DF

 ; ------------------------------------
 ; PROPERTIES INDEPENDENT OF TIME
 ; ------------------------------------

TIME , , ,   .1000E+01   .1000E+01 N
NUM  -1  -1  -1  -1  -1
CONS   .3500E+00   .8000E+00   .6000E+00   .10000E+04  .1013250E+06
CONV ABS   .1000E-05   .1000E-05   .1000E-05   .1000E-02   .1000E-02
CONV REL   .1000E-05   .1000E-05   .1000E-05   .1000E-02   .1000E-02
PARMS  2 N N
PATH
FILES,OUT1,OUTFULL,PLOTF
REPORT  Y N N Y Y N
OPTIONS N,, Y,, N
TITLE DUMP   DUMP TITLE
TITLE OUTPUT OUTPUT TITLE
TITLE PLOT   PLOT TITLE

 ; FIRE PROPERTIES

FIRE   1   1   .000000E+00   .250000E+06

 ; ROOM DIMENSIONS

ROOM  1   .0000E+00   .3000E+01   .2000E+02

AMB  1   .0000E+00   .0000E+00   .1200E+01
AMB  1 ,,,,

 ; VENT DIMENSIONS

VENT  1  1 -1   .20000E+01   .00000E+00   .20000E+01
```

NIST-114A
(REV. 3-90)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

BIBLIOGRAPHIC DATA SHEET

1. PUBLICATION OR REPORT NUMBER
NISTIR 4343

2. PERFORMING ORGANIZATION REPORT NUMBER

3. PUBLICATION DATE
July 1990

4. TITLE AND SUBTITLE

The Consolidated Compartment Fire Model (CCFM) Computer Code Application CCFM.VENTS - Part II: Software Reference Guide

5. AUTHOR(S)

Glenn P. Forney and Leonard Y. Cooper

6. PERFORMING ORGANIZATION (IF JOINT OR OTHER THAN NIST, SEE INSTRUCTIONS)

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
GAITHERSBURG, MD 20899

7. CONTRACT/GRANT NUMBER

8. TYPE OF REPORT AND PERIOD COVERED

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

Sponsored in part by:
Naval Sea Systems Command
Department of the Navy
Washington, DC 20362

10. SUPPLEMENTARY NOTES

11. ABSTRACT (A 200-WORD OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, MENTION IT HERE.)

A project was carried out at The National Institute of Standards and Technology (NIST) to study the feasibility of developing a new-generation, multi-room, compartment fire model computer code, called the Consolidated Compartment Fire Model (CCFM) computer code. The idea was that such a code would consolidate past progress in zone-type compartment fire modeling, and allow readily for integration of future advances with the greatest possible flexibility. The project lead to the development of a prototype multi-room CCFM product called CCFM.VENTS.

This is Part II of a four-part report which documents the above effort. The main objective of this Part II document is to document the design and underlying structure of the CCFM.VENTS computer software. It serves as a guide for those persons interesting in extending, modifying and if necessary correcting CCFM.VENTS at a later date.

The other three parts of this report are: Part I: Technical Basis; Part III: Catalog of Algorithms and Subroutines; and Part IV: User Reference Guide.

12. KEY WORDS (6 TO 12 ENTRIES; ALPHABETICAL ORDER; CAPITALIZE ONLY PROPER NAMES; AND SEPARATE KEY WORDS BY SEMICOLONS)

building fires; compartment fires; compuer models; fire models; mathematical models; vents; zone models

13. AVAILABILITY

□ UNLIMITED
X
□ FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NATIONAL TECHNICAL INFORMATION SERVICE (NTIS).
□ ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GOVERNMENT PRINTING OFFICE, WASHINGTON, DC 20402.
□ ORDER FROM NATIONAL TECHNICAL INFORMATION SERVICE (NTIS), SPRINGFIELD, VA 22161.
X

14. NUMBER OF PRINTED PAGES
92

15. PRICE
A05

ELECTRONIC FORM