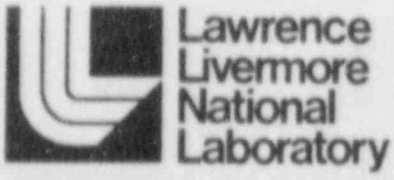
NUREG/CR-3686
UCRL-15597 Vol. 3

WIPS—Computer Code for Whip and Impact Analysis of Piping Systems

Part C—Programmer's Manual

G. H. Powell, J. P. Hollings, D. G. Row, P. Chen, F-C. Hu, M. Mahasuverachai, B. Mosaddad, P. Nicklin, S. Nour-Omid, C. Oughourlian, and A. Riahi;
University of California, Berkeley, CA

Prepared for
U.S. Nuclear Regulatory Commission



GPO Printed copy price: $4.75

NUREG/CR-3686
UCRL-15597 Vol. 3
Intramural #3371609

WIPS—Computer Code for Whip and Impact Analysis of Piping Systems

Part C – Programmer's Manual

Manuscript Completed: March 1983
Date Published: June 1984

Prepared by
C. H. Powell, J. P. Hollings, D. G. Row, P. Chen, F-C. Hu, M. Mahasuverachai, B. Mosaddad, P. Nicklin, S. Nour-Omid, C. Oughourlian, and A. Riahi;
University of California, Berkeley, CA

Lawrence Livermore National Laboratory
7000 East Avenue
Livermore, CA 94550

Prepared for
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, D.C. 20555
NRC FIN No. A0136

WIPS - COMPUTER CODE FOR WHIP AND IMPACT ANALYSIS OF PIPING SYSTEMS

PART C

PROGRAMMER'S MANUAL

by

G. H. Powell, J. P. Hollings, D. G. Row, P. Chen, F-C. Hu, M. Mahasuverachai, B. Mosaddad, P. Nicklin, S. Nour-Omid, C. Oughourlian, and A. Riahi

University of California,
Berkeley, California

Prepared for
Lawrence Livermore National Laboratory
Livermore, California
under LLL Subcontract No. 3371609

ACKNOWLEDGEMENTS

The authors wish to acknowledge the assistance and patience of Program Managers P. Albrecht, M. Vagins, and G. Weidenhamer of the Nuclear Regulatory Commission and P. Smith, C-K. Chou, and T-Y. Chuang of the Lawrence Livermore National Laboratory.

Special acknowledgement is due to R. Chun, Lawrence Livermore National Laboratory, for his invaluable assistance in running examples on the CRAY computer and for his extremely thorough review of the manuscript.

L. Calvin extended herself above and beyond the call of duty to prepare the manuscript. The good figures were drafted by G. Feazell.

WIPS - COMPUTER CODE FOR WHIP AND IMPACT ANALYSIS OF PIPING SYSTEMS

PART C

PROGRAMMER'S MANUAL

ABSTRACT

This report presents information for use by programmers who may wish to add new features to WIPS (particularly new element types). Information is provided on the free form data input package; the data base manager used in WIPS-ANAL; the procedures required to add new element types to WIPS-ANAL and to the WIPS-INPT package; and the file structures in WIPS-INPT. Information is also provided on computer system requirements.

TABLE OF CONTENTS

C1. SYSTEM REQUIREMENTS

SUMMARY

This section identifies the machine-dependent features of the WIPS code as well as general computer system requirements. Section C1.1 is concerned mainly with the VAX-UNIX system which was used for code development and testing. Sections may need to be added for other computer systems.

CONTENTS

C1.1 GENERAL REQUIREMENTS

C1.1.1 PROGRAMMING LANGUAGE

All WIPS modules, except WIPS-ANAL, execute in interactive (foreground) mode. These modules are coded in FORTRAN-77 and are heavily dependent on FORTRAN-77 features. They would be very difficult to modify to execute under FORTRAN-66.

The WIPS-ANAL module has been developed and tested using FORTRAN-77 but makes only limited use of FORTRAN-77 features, specifically for file opening and closing. WIPS-ANAL also uses one random-access file. This module could be modified without major difficulty to execute under FORTRAN-66.

C1.1.2 CORE REQUIREMENTS

All WIPS modules, except WIPS-ANAL, make use of a simple in-core data base manager (DBM) (see Section C3.1). Except for a few labelled COMMONS and short dimensioned arrays, data is stored by the DBM in blank common (COMMON L(...)). The length of array L is set in the main program of each module. The length required depends on the size of problem to be considered. In the basic version of WIPS, the length of L is 10000 in WIPS-MODL, 50000 in WIPS-RSLT, and 5000 in all other modules (except WIPS-ANAL). These lengths should be adequate for both small and large problems. If the length is inadequate, a message will be displayed by the DBM during execution.

The WIPS-ANAL module makes use of a more sophisticated DBM (see Section C3.2). This DBM operates in-core if adequate blank COMMON length is dimensioned, with automatic overflow to a single random access file (the PAUS file) if necessary.

For the VAX-UNIX virtual memory system used to develop WIPS, computational efficiency suffers substantially when the random access file is used. A blank COMMON length of 2 megabytes (i.e. COMMON L (500000)) has been used for most analyses on this system. This permits "in-core" execution for problems of small and intermediate size (the system has 4 megabytes of core but in a multi-user environment the core allocated per user is much less and virtual memory paging occurs) but not for large problems involving straight pipe, elbow, or slab substructures. On other systems, experimentation will be needed to determine the most appropriate blank COMMON lengths.

C1.1.3 WIPS-ANAL EXECUTION

C1.1.3.1 Mini Versus Mainframe

All WIPS modules, except WIPS-ANAL, require only small execution times, and it is appropriate to execute them on a mini-computer. WIPS-ANAL requires substantially longer execution times, ranging from a few minutes (for small piping models using *beam* or *pipe* elements) through a few hours (for larger models using *beam* or *pipe* elements) up to tens or hundreds of hours (for models incorporating straight pipe, elbow, or slab substructures).

It is appropriate to execute smaller problems on a mini-computer, but large problems can be executed more efficiently using a large scientific mainframe. This could be done by preparing the WIPS-ANAL files (DATA, ECHO, SLOG, RSLT, and, if needed, PAUS and PAUZ) on a mini-computer, transferring these files to a mainframe for execution of WIPS-ANAL, then transferring them back for post-processing.

C1.1.3.2 ANAL Execution on VAX/UNIX System

Execution of WIPS-ANAL on the VAX/UNIX system at the University of California, Berkeley, is initiated from WIPS-EXEC via a "C" program a.sh. This program questions the user on the cpu time limit and whether background or batch execution is required. For background execution, the program uses the system program "limit" to assign the cpu limit, file size

limit (20 Mb), and coredump file limit (5 Mb), then begins execution. For batch execution, it queries the user on whether day or night priority is to be assigned, then creates the UNIX command:

batch -cpu=**** -file=10m -core=5m -night -anal.sh

where **** = cpu time limit, and for day priority "-night" is omitted. The anal.sh file is a command shell script which simply contains the command "anal".

C1.1.3.3 ANAL Execution Stand-Alone

Although it is convenient to initiate WIPS-ANAL execution from WIPS-EXEC, this requires a system-dependent procedure of the type described in the preceding section. A simpler procedure is to execute WIPS-ANAL as a stand-alone program. The procedure is as follows.

(1) Create a file "wipsdat" containing only the filename of the DATA file to be used, in A8 format (e.g. DATA0102).

(2) Assign appropriate cpu and file size limits.

(3) Begin execution of ANAL.

C1.1.4 OTHER SYSTEM DEPENDENT FEATURES

C1.1.4.1 WIPS-EXEC

The WIPS-EXEC module has a small number of system dependent features. These are explained in Section C5.1.4.

C1.1.4.2 WIPS-ANAL Execution

In the basic version of WIPS, the WIPS-ANAL module is executed by a *program* call from WIPS-EXEC. In general it may be preferable to execute WIPS-ANAL as a separate program (see Section C1.1.3).

C1.1.4.3 Overlay

The VAX/UNIX system allows one program to be executed from another by means of a "PROGRAM" call. Each WIPS module is thus set up as a separate program.

On systems which do not have this feature, it will be necessary to set up the modules as overlays. WIPS-EXEC will be the root overlay, and each WIPS module will be a level 1 overlay. The "CALL PROGRAM" statements in WIPS-EXEC must be replaced with "CALL OVERLAY" statements. WIPS-EXEC does not make use of blank COMMON.

C1.1.4.4 New Line Suppression

For interactive input, the sign "$" at the end of a FORMAT statement is used to force continuation of the current line. This is not standard FORTRAN-77.

C1.1.4.5 Plot Package

The plotting routines used by WIPS-GEOM and WIPS-RSLT are in FORTRAN-77 and should not be system dependent. These routines apply for the Tektronix 4662 pen plotter, 4013 and 4015 graphics terminals, and 4027 color graphics terminal. Provisions for CALCOMP plotting have been included but not tested.

The plot package consists of a number of primitive routines for pen move, drawing of lines, and plotting of text. The records transmitted to the plot devices are automatically padded with nonexecutable data to compensate for data transmission rates. Control information for plotting is set in SUBROUTINE DEVICE.

C1.2 VAX/UNIX SYSTEM AT U.C. BERKELEY

WIPS has been developed on the DEC VAX 11/780 at the University of California, Berkeley. The configuration of this system is as follows:

Operating System:	UNIX 7th edition, Virtual VAX-11 Version, Berkeley Software Distribution 4.0.
Core Capacity:	4 Mbytes
Maximum Process Size:	16 Mbytes
Disk Capacity:	141 Mbytes per filesystem
Number of Tape Drives:	2 (not used by WIPS)

C2. FREEFORM INPUT PACKAGE

SUMMARY

WIPS accepts lines of freeform data interactively and checks the data for syntax and data sequence errors. This section describes the subroutines which make up the data input package.

CONTENTS

C2.1 DATA FORMAT

The WIPS freeform input routine is for data input in an interactive mode. The routine interprets an input data line in the form:

d1/d2/.../di/.../dn/O1=e1/O2=e2/.../Oi=ei/.../Om=em// optional comment

where

di = "positional" data item, identified by position in line. May be an integer, real, or alpha word.

ei = optional data item, identified by option code Oi. May be an integer, real, or alpha word.

Oi = option code (alpha, 1 to 4 characters).

/ = separator, consisting of a comma and/or one or more blanks (*not* a slash).

// = a specified number of consecutive blanks (currently 3), indicating end of data.

n = number of positional data items.

m = number of optional data items.

The optional comment is for the convenience of the user and is ignored.

Example:

TYP1,3,7.5,6.E-3,A4GX,PROP=3,DX=2.2E8,LDIS=YES EXAMPLE LINE

The data may be any sequence of integer, real, and/or alpha words, as defined by the programmer. Data of the following modes may be interpreted:

(a) Integer (I): integer numbers only. No decimals or blanks.

(b) Real (R): any FORTRAN I, F, or E conversion.

(c) Alpha (A): 1 to 4 alphanumeric characters. No blanks.

Two consecutive commas (,,) denote a zero number or a blank word.

A single input line may be interpreted in a single pass or in a series of passes. A maximum of 10 words may be interpreted from a single data line.

For each line to be interpreted, one call must be made to subroutine DLIMIT, followed by one or more calls to subroutine FREEF. In addition, subroutines, GETLIN, SETFOR, SETOPT, MESSG, and YESNO may be called. Subroutine HELP must be provided by the programmer.

C2.2 SUBROUTINE PACKAGE

C2.2.1 SUBROUTINE DLIMIT

C2.2.1.1 Argument List

The subroutine statement is as follows:

SUBROUTINE DLIMIT (IW,NWORDS,IHELP,ICOL,MAXB)

i. which

(a) IW(80) = character*1 array which holds the data line to be interpreted, in 80A1 format. Must be set before entry to DLIMIT.

(b) NWORDS = integer variable or constant, specifying number of words to be interpreted. Must be set before entry to DLIMIT. Must not exceed 10.

(c) IHELP = help code. Must be set before entry to DLIMIT. Must be an integer variable, not a constant, because its value may be changed in DLIMIT. If the first word in the data line is "help," the following statements are executed in DLIMIT.

```
CALL HELP (IHELP)
IHELP = -IHELP
RETURN
```

Subroutine HELP must be provided, to print an appropriate guidance message. If a negative value is returned for IHELP, ask for the data line to be re-entered.

(d) ICOL(2,N), where N > NWORDS, = integer array for return of delimiting information, as explained in the following section.

C2.2.1.2 Purpose

Subroutine DLIMIT breaks the data line into groups of columns, each group defining a word. The numbers of the first and last columns of word I are stored in ICOL(1,I) and ICOL(2,I). The ICOL array is used directly by subroutine FREEF and must not be changed.

A zero value for ICOL(1,I) indicates a zero or blank word. If the limit of MAXB consecutive blanks is reached before NWORDS groups of columns have been delimited, ICOL(1,I) is set to zero for the remaining words.

C2.2.2 SUBROUTINE FREEF

C2.2.2.1 Argument List

The subroutine statement is as follows:

SUBROUTINE FREEF (IW,NWORDS,IFORM,LTITLE,IERR,INTW,ALPHW,IAI, REAW,ICOL,IOPT,NOPT,IGNO)

in which

(a) IW(80) = character*1 array holding data line, as before.

(b) NWORDS = number of words to be considered in this pass through FREEF (see array IFORM).

(c) IFORM(N), where N > NWORDS, = integer array defining data modes, such that IFORM(I) defines the data mode for word I, as follows:

0 = ignore

2 = real (R) data

3 = optional data (identified by an option code)

-n = alpha (A) word of n characters (max. 4). If more than n characters are input, the first n characters are extracted and the remainder are ignored. If fewer than n characters are input, trailing blanks are added.

Array iform must be set before entry to FREEF, either by means of a DATA statement or by calling subroutine SETFOR.

(d) LTITLE = number of characters in the printed message requesting input of the data line. For example, if the message is

ENTER COORDS X,Y,Z:

then LTITLE = 19. This is used for pointing to errors in the data line. If character I in the data line is incorrect, FREEF prints an error message and points to character (I+LTITLE) in the line displayed on the screen.

(e) IERR = error code, returned by FREEF. If IERR = 0, no error has been found; otherwise the data line contains an error, and the user must be requested to re-enter the line.

(f) INTW = one-dimensional integer array in which integer words are returned. See Section E12.3.2 for locations in INTW which are used.

(g) ALPHW = one-dimensional character*4 array in which alpha words are returned (see Section E12.3.2).

(h) IAI = storage code for INTW and IALPH arrays. Specify zero if INTW and ALPHW occupy separate storage. Specify 1 if INTW and ALPHW have the same first word address. See Section E12.3.2.

(i) REAW = one-dimensional real array in which real words are returned. See Section E12.3.2.

(j) ICOL(2,N) = delimiting array set by subroutine DLIMIT.

(k) IOPT (4,NOPT) = mixed alpha-integer array defining optional data codes which may be used. Column I of IOPT contains information for option code I, as follows:

IOPT(1,I) = alpha word (A4) defining option code (e.g. "prop")

IOPT(2,I) = acceptable abbreviation (e.g. "pr")

IOPT(3,I) = data mode (1=I; 2=R; -n=An)

IOPT(4,I) = location in corresponding data array (INTW for I; REAW for R; ALPHW for A) in which data is to be placed (e.g. if IOPT(2,I) = 1 and IOPT(3,I) = 4, integer data is to be extracted and placed in INTW (4)). See Section E12.3.2.

Array iopt must be set before entry to FREEF, either by means of a DATA statement or by calling subroutine SETOPT.

(l) IGNO = ignore code for optional data. For each optional data word (IFORM(I) = 3), the option code is compared with all codes in array IOPT. If a match is not obtained and IGNO = 0, the data word is ignored. If a match is not obtained and IGNO = 1, an error message is printed (incorrect option) and IERR is set to 1.

C2.2.2.2 Return of Data Items

Subroutine FREEF extracts data items from the data line and returns them in arrays INTW, ALPHW, and REAW.

The data modes are defined by arrays IFORM and IOPT. Positional items (1, 2, or -n in array IFORM) are stored in the corresponding array in the order in which they appear in the line (i.e. first integer word goes in INTW(1), first real word in REAW(1), etc.). The optional items are placed in the corresponding array in the location defined by IOPT(3,I). This is necessary because optional data items need not appear in any specified sequence.

If the storage code, IAI, is set to 1, then INTW and ALPHW are assumed to share the same storage locations. In this case, positional integer and alpha items are both counted to determine their storage locations. For example, if the format array, IFORM,, is (1,-4,1), the first integer item is stored in INTW(1), the alpha item in ALPHW(2), and the second integer item in INTW(3).

The storage code, IAI, is *not* considered by FREEF in storing optional data items. If INTW and ALPHW share the same storage locations, this must be taken into account in setting up array IOPT.

C2.2.2.3 Default Values for Optional Data Items

If IGNO = 1, FREEF flags improper options and sets IERR = 1. Except for this, however, FREEF does not indicate whether or not any particular option code has been used. The programmer must therefore devise means to ensure that default values are used if a particular option is not exercised. One way is to initialize the appropriate locations in INTW, ALPHW, and REAW to the default value before calling FREEF. Then, if any option is not exercised, the default value is returned.

C2.2.3 SUBROUTINE GETLIN

The subroutine statement is as follows:

SUBROUTINE GETLIN (IW)

in which IW(80) is a character*1 array holding the data line, as before. The subroutine reads a data line.

In many applications, the function of GETLIN can be performed by a read statement. In WIPS, however, GETLIN also writes the data line to the WIPSLOG file so that a log of the session can be printed.

C2.2.4 SUBROUTINE SETFOR

The subroutine statement is as follows.

SUBROUTINE SETFOR (IFORM,I1,I2,I3,I4,I5,I6,I7,I8,I9,I10)

in which

(a) IFORM(10) = integer array being set up to define data modes.

(b) I1,I2,etc. = integers defining data modes for up to 10 data items (1, 2, 3, or -n).

Subroutine SETFOR simply puts I1, I2, etc. in array IFORM.

C2.2.5 SUBROUTINE SETOPT

The subroutine statement is as follows:

SUBROUTINE SETOPT (IOPT,NC,NAM,NAMA,MOD,LOC)

where

(a) IOPT(4,M) = integer array defining data codes (M = number of columns in array).

(b) NC = column in IOPT to be set by this call.

(c) NAM = A4 alpha word (e.g. "prop"). This is put in IOPT(1,NC).

(d) NAMA = acceptable abbreviation (e.g. "pr"). This is put in IOPT(2,NC).

(e) MOD = data mode (1, 2, 3, or -n). This is put in IOPT(3,NC).

(f) LOC = location in INTW, ALPHW, or REAW array. This is put in IOPT(4,NC).

C2.2.6 SUBROUTINE MESSG

Subroutine MESSG may be called to print a user prompt or any other message. The subroutine statement is as follows:

SUBROUTINE MESSG (MM,LM,NS)

in which

(a) MM(80) = character*1 array containing the message to be printed.

(b) LM = message length (number of characters), which is returned by MESSG (this avoids counting the title length by hand for the call to FREEF).

(c) NS = number of blank lines to be printed before message.

In the call statement, the message may be of any length up to 80 characters. The last character must be "$". If a carriage return is required after the message is displayed, the second-to-last character must be "/". The message is both displayed and written in the WIPSLOG file.

Examples:

CALL MESSG (' ENTER COORDS X,Y,Z, : $' ,LTITLE,0)

CALL MESSG (' ENTER VALUES AS FOLLOWS /$' ,K,2)

C2.2.7 SUBROUTINE COMMNT

Subroutine COMMNT may be called to print a command or explanatory message. The subroutine statement is as follows:

SUBROUTINE COMMNT (MM)

in which MM(80) = character*1 array containing the comment to be printed. The last character must be "$". The comment is both displayed and written in the WIPSLOG file.

Example: The statement

CALL COMMNT (' ENTER VALUES AS FOLLOWS $')

C2.2.8 SUBROUTINE ERROR

Subroutine ERROR may be called to print an error message. The subroutine statement is as follows:

SUBROUTINE ERROR (MM)

in which MM(40) is character*1 array containing the error message. The last character must be "$".

Example: The statement

CALL ERROR (' INCORRECT C.P. TYPE $')

will display the following line and also write it in the WIPSLOG file:

***ERROR - INCORRECT C.P. TYPE

C2.2.9 SUBROUTINE YESNO

C2.2.9.1 Argument List

Subroutine YESNO may be called to print a question and detect a "yes" or "no" reply. The subroutine statement is as follows:

SUBROUTINE YESNO (MM,IY,NS)

in which

(a) MM(80) = character*1 array containing the question to be printed.

(b) IY = reply code returned by YESNO (1 = "yes"; zero = "no").

(c) NS = number of blank lines to be printed before message.

In the call statement, the message may be any length up to 80 characters. The last character must be "$". The message is both displayed and written in the WIPSLOG file.

Example:

```
CALL YESNO (' LAST POINT  ?  :  $' ,IYN,2)
```

C2.2.9.2 Acceptable Responses

A "yes" or "y" response is interpreted as "yes". A "no", "n", or blank response is interpreted as "no". For any other response except "help", the message "eh?" is displayed, and the response must be re-entered. For a "help" response, the following call is made to subroutine HELP:

```
CALL HELP (IY)
```

After the return from HELP, the question is repeated. If a help message is to be printed, the value of IY must be set before calling YESNO.

C2.2.10 SUBROUTINE HELP

Subroutine HELP must be provided by the programmer. The subroutine statement is as follows:

```
SUBROUTINE HELP (IHELP)
```

in which IHELP = integer which identifies the message to be printed.

The value of IHELP must be set by the programmer in the calls to DLIMIT and/or YESNO. The purpose of the subroutine is to print an appropriate message, then return.

C3. DATA BASE MANAGERS

SUMMARY

WIPS makes use of two different data base packages, namely (1) a simple, in-core, sequential data base (which is used by all modules except WIPS-ANAL) and (2) a sophisticated hierarchical data base designed for finite element analysis (which is used by WIPS-ANAL). The data structures and the programming procedures for use of the two data base managers are described in Sections C3.1 and C3.2, respectively.

CONTENTS

C3.1 SIMPLE DATA BASE MANAGER

C3.1.1 GENERAL

All of the WIPS modules, except WIPS-ANAL, have modest storage demands and execute in-core. Small amounts of data are stored in labeled COMMON storage and in arrays with fixed dimensions. Most data is stored in blank COMMON, with space allocation controlled by a simple data base manager (DBM). The DBM was developed jointly by Structural Software Development, Inc., Berkeley, California, and Professor E. L. Wilson of the University of California at Berkeley.

The DBM is simple to use and effective for small problems, but it is not efficient for problems involving many arrays and large amounts of data. It includes provisions for storage of data both in-core and on auxiliary storage. In the existing WIPS modules, only the in-core options have been used.

C3.1.2 DATA AND DESCRIPTOR ARRAYS

Data arrays are stored sequentially, in compacted form, in blank COMMON (array L(MTOT)). Each array is preceded by a five-word descriptor array (ID(5)). For an in-core data array beginning at L(NA), the descriptor array contains the following:

ID (1) = L(NA-5) = Number of data words in array.

ID (2) = L(NA-4) = Precision (number of storage words per data word).

ID (3) = L(NA-3) = Number of columns in array.

ID (4) = L(NA-2) = Zero (= in-core).

ID (5) = L(NA-1) = Array name (alphanumeric identifier, maximum 4 characters beginning with an alphabetic character).

where NA = first word address (fwa) of data array in L.

If an array is filed on auxiliary storage, the data and descriptor arrays are transferred to the file. The descriptor is retained in-core in a modified form, and the data array is replaced by an additional four-word descriptor array (IID(4)). The modified descriptor is as follows:

ID (1) = 4 (size of additional descriptor)

ID (2) = 1 (precision of additional descriptor)

ID (3) = 1

ID (4) = Unit number on which array is stored.

ID (5) = Array name.

The additional descriptor is as follows:

IID (1) = Size of data array.

IID (2) = Precision of data array.

IID (3) = Number of columns in data array.

IID (4) = Record number on file.

C3.1.3 OPERATIONS

C3.1.3.1 General

The DBM is used by making subroutine calls of a variety of types, as described in the following sections. Arguments marked by asterisks are returned by the subroutines.

C3.1.3.2 DEFINE - Define New Array

CALL DEFINE (NAME,LOC*,NROWS,NCOLS,IPREC)

where

NAME = alphanumeric array name

LOC = fwa in L assigned to data array

NROWS = number of rows in data array

NCOLS = number of columns in data array

IPREC = precision code (1 = single; 2 = double)

New arrays are added at the end of the data base.

C3.1.3.3 LOCATE - Locate Array

CALL LOCATE (NAME,LOC*,NROWS*,NCOLS*)

where

NAME = array name

LOC = fwa in L

NROWS = number of rows

NCOLS = number of columns

C3.1.3.4 DELETE - Delete Array

CALL DELETE (NAME)

where

NAME = array name

The descriptor and data arrays are deleted, and blank COMMON is compacted. Hence, array locations may be changed.

C3.1.3.5 EXPAND - Expand Array

CALL EXPAND (NAME,LOC*,NROWS,NCOLS)

where

NAME = array name

LOC = new fwa in L

NROWS = new number of rows (not less than existing number)

NCOLS = new number of columns (not less than existing number)

EXPAND involves deletion of the current array and relocation to the end of the data base. Hence, array locations may be changed. The DBM, as currently implemented, contains an error and does *not* allow the number of rows to be changed.

C3.1.3.6 FILE - Save on File

CALL FILE (NAME,IUNIT)

where

NAME = array name

IUNIT = unit number

Arrays are stored sequentially on each file, one record per array.

C3.1.3.7 RECALL - Recall from File

CALL RECALL (NAME)

where

NAME = array name

C3.1.3.8 GETPKT = Get Column of Data from an Array

CALL GETPKT (DATA,NAME,NPKT)

where

DATA = array where data in column is to be placed

NAME = array name

NPKT = packet (column) number to be extracted

C3.1.3.9 PUTPKT - Put Column of Data in an Array

CALL PUTPKT (DATA,NAME,NPKT)

where

DATA = array from which data is to be taken

NAME = array name

NPKT = packet (column) number where data is to be placed

C3.1.3.10 DELPKT - Delete Column and Compact Array

CALL DELPKT (NAME,NPKT)

where

NAME = array name

NPKT = column number to be deleted

The array is not relocated. All columns beyond the deleted column are moved back one column.

C3.1.3.11 INSPKT - Insert Column and Expand Array

CALL INSPKT (DATA,NAME,NPKT)

where

DATA = array containing data to be inserted

NAME = array name

NPKT = column number where data is to be inserted

The array is automatically expanded, and hence also relocated. The columns beyond the added column are all moved forward one column.

C3.1.3.12 LOCPKT - Locate Column Containing a Data Word

CALL LOCPKT (NAME,NROW,IDATA,NPKT*)

where

NAME = array name

NROW = row to be searched

IDATA = single word (typically integer or alpha) for which match is required

NPKT = number of first column with a matching word in the specified row

C3.1.3.13 LOCPK2 - Locate Column Containing Two Data Words

CALL LOCPK2 (NAME,NROW1,NROW2,IDATA1,IDATA2,NPKT*)

As for LOCPKT, but match two data words.

C3.1.3.14 PRTARR - Print Array

CALL PRTARR (NAME)

where

NAME = array name

C3.1.4 INITIALIZATION

The following common blocks must be provided for by the host program, initialized, and reserved solely for use by the DBM.

COMMON/DBMS/MTOT,NARA,NSIZ,NDIR

MTOT - Initialize to size of blank common.

NARA - Initialize to zero.

NSIZ - Initialize to 1.

NDIR - Initialize to 5.

COMMON/TAPE/MINP,MOUT,NTP(10),NRP(10),NTR(10)

MINP - Initialize to input unit number (typically 5).

MOUT - Initialize to output unit number (typically 6).

NTP(10) - Initialize to acceptable scratch or permanent file unit numbers.

NRP(10) - Initialize to 1.

NTR(10) - Initialize to 1.

COMMON L (MTOT)

C3.2 WIPS-ANAL DATA BASE MANAGER

C3.2.1 GENERAL

WIPS-ANAL incorporates a data base manager (DBM) specifically designed for large capacity finite element analysis. During WIPS-ANAL execution, data is stored in blank COMMON (the *corepool)* and (if necessary) on a random access file (the *auxiliary storage pool).* At a pause in the analysis, blank COMMON is written to the PAUZ file. The random access file is the PAUS file. With a virtual memory operating system, data in blank COMMON is not necessarily in physical core at any given time.

C3.2.2 DATA STRUCTURE

C3.2.2.1 Records, Arrays, and Systems

The DBM uses a three-level hierarchical data structure. The levels in the hierarchy are as follows.

(1) RECORD: A record is the smallest module handled by the DBM. Typically it is a subset of data pertaining to a particular entity. Each record contains actual data and is identified by a sequence number.

(2) ARRAY: An array is a collection of one or more records which relate to a common entity. Each array contains a directory of records and is identified by an alphanumeric identifier. An array should not be confused with a FORTRAN data array.

(3) SYSTEM: A system is a collection of one or more related arrays. Each system contains a directory of arrays and is identified by an alphanumeric identifier.

A request for a specific data module accesses a *description packet.* Each description packet contains information for recovery of the data module (its size, location, auxiliary storage parameters, etc.). The hierarchal nature of the data requires that the description packets be organized into a series of dependent directories. For example, a system has associated with it a dependent array directory, consisting of a number of array description packets. The directories are contained in three tables of adjustable length, one table for systems, one for arrays, and one for records.

C3.2.3 DBM STRUCTURE

The DBM consists of four components managers, namely (1) the Directory Manager, (2) the Corepool Manager, (3) the I/O Manager, and (4) the Command Interpreter. All DBM subroutine names start with the characters "MN". Programs using the DBM should not use subroutine, function, or common block names beginning with "MN".

The tasks performed by the component managers are as follows. The Directory Manager handles all data requests. Its primary purpose is the maintenance and control of indices to the individual data structures. This includes creation, deletion, compaction, and recovery. The Corepool Manager controls in-core storage allocation. It maintains a list of active data records and controls allocation, deletion, and compaction of these records. The I/O Manager handles all transfer of data between the corepool and the auxiliary storage pool. The Command Interpreter is the interface between the program and the DBM. It interprets commands to the DBM and translates them into operations to be performed by the DBM.

C3.2.4 INTERFACE COMMANDS

C3.2.4.1 General

The DBM is accessed by calls to a number of interface subroutines. The call statement has the form:

CALL AAABBB (...)

in which AAA represents the operation to be performed and BBB the type of data to be operated on (record, array, etc.).

C3.2.4.2 Basic Commands

The basic commands are as follows.

DEF*** Defines the size, nature, and other parameters for a new data module. If a module has already been defined, the call serves to alter its parameters.

DEL*** Deletes all storage and references to the specified data module. A data module cannot be accessed after a delete.

ACT*** Activates the specified data module to allow processing of its contents. If the data is not currently in core, a transfer from auxiliary storage to core is performed.

DEA*** Deactivates the specified data module. Deactivation allows core storage presently occupied by the module to be reallocated. If reallocation is actually performed, the data module is written to auxiliary storage before the core space is freed.

GET*** Transfers the specified data module from auxiliary storage to a specified location in core.

PUT*** Transfers the specified data module from core to auxiliary storage.

C3.2.4.3 Alphanumeric Identifiers

Arrays and systems are identified by alphanumeric words. Identifier words may have a maximum of four characters, the first of which must be an alphabetic character or blank. The locations of blanks are important (i.e. 4H_ABC and 4HABC_ are not the same).

C3.2.4.4 Activity Status

A record, array, or system may be *undefined, inactive, active*. A data module is undefined if it has never been named in a DEF*** command or if it has been deleted by a DEL*** command. When it is first defined, a data module is initially inactive. It becomes active when named in an ACT*** command and inactive when named in a DEA*** command. An active data module is "ready for action", whereas an inactive array is "waiting for action".

C3.2.4.5 Error Trace

In any subroutine which calls the DBM, the programmer is required to identify the calling subroutine and the locations in the subroutine from which the DBM calls are made. The procedure is illustrated by the following example.

```
      SUBROUTINE ADDLOD

      CALL PRONAM (6HADDLOG)

  100 CALL ACTSYS (4HLOAD,100)

  150 CALL DEFARR (4HLOAD,4HGRAV,1,NEQ,1,0,150)
```

The call to PRONAM makes the current subroutine name available to the DBM. The line number and corresponding value in the argument list then identify the location of each subsequent subroutine call. In this example, if an error were detected in ACTSYS, the following message might be printed:

***FATAL ERROR - (12) DETECTED BY MNACTS AT STATEMENT NUMBER 100 IN ADDLOD**

Subroutine MNACTS is the DBM routine which detected the error. A description of the errors is given in Section C3.2.7.

C3.2.5 OTHER DATA TYPES

C3.2.5.1 Files and Packets

In addition to records, systems, and arrays, the DBM recognizes data FILES. A file is at the same level as an array. However, whereas an array may contain records of different length, a file must consist of a number of equal length PACKETS. Each packet can contain any mixture of real, integer, and alpha data. Physically, the packets making up a file are stored as a number of records, typically with several files per record. The number of packets per record must be specified when the file is defined (DEFFIL command) and also the number of records to be assigned initially to the file. If the total number of packets is known in advance, the required number of records can be specified exactly. If the number of packets is not known, the initial number should be a reasonable lower bound estimate. If this initial number is exceeded, the DBM automatically increases the number of records.

Each file is automatically assigned a SLOT number at the time it is defined. Subsequent GETFIL and PUTFIL commands must then reference not the alphanumeric file name but the numeric slot number. This is done for computational efficiency to allow files to be located quickly by the DBM without the directory search needed to locate an array.

The GETFIL and PUTFIL commands allow a specific packet to be extracted from a file. Typically a large number of these commands will be used during program execution to gain access to many packets (for example, if a file contains node coordinates, the coordinates for a single node will constitute a packet and access to the coordinates will typically be needed many times). The GETFIL command copies a packet to a "reserved" location in core, typically a labelled COMMON block or a dimensioned array. If the data in the packet is modified, the PUTFIL command returns the packet to the file. Note that GETFIL and PUTFIL commands typically will transfer data from one core location to another, not between core and auxiliary storage.

C3.2.5.2 Buffers

Core storage can be "reserved" by setting up labelled COMMON blocks or dimensioned arrays. Storage can also be reserved by activating a BUFFER within the corepool. The ACTBUF command reserves a specified amount of core for a *buffer* and assigns a slot number to the buffer. The first word address assigned to the buffer can be obtained using the LOCBUF command. The reserved area can then be used in any way desired by the programmer. The programmer must take care not to use more than the reserved storage area.

Labelled COMMON blocks will typically be used to reserve storage for short or well-defined data packets. Buffers will typically be used if the amount of storage to be reserved is large or is known only at execution time (for example, storage for a stiffness matrix).

C3.2.5.3 Caution in Use of LOC*** Commands

The DBM automatically compacts core storage if the total requested storage exceeds the blank COMMON length. This process involves transferring low priority arrays and records to auxiliary storage to release in-core space. The compaction will change the core locations of some or all of the data modules which remain in core. This requires that the programmer exercise care in using LOC*** commands. Consider, for example, the following sequence.

```
10 CALL ACTBUF (IBUF,ISIZ,10)
20 ILOC = LOCBUF (IBUF,20)
60 CALL ACTBUF (JBUF,JSIZ,60)
70 JLOC = LOCBUF (JBUF,70)
```

Statement 60 may initiate a compaction of core storage, which could affect the location of IBUF. Because ILOC was set prior to the compaction, it would no longer be the first word address of IBUF.

The following procedure is thus recommended:

(1) Activate all required arrays, buffers, etc.

(2) Locate all data.

For the above example, the following coding would be valid:

```
10 CALL ACTBUF (IBUF,ISIZ,10)
60 CALL ACTBUF (JBUF,JSIZ,60)
20 ILOC = LOCBUF (IBUF,20)
70 JLOC = LOCBUF (JBUF,70)
```

C3.2.6 COMMAND FORMATS

C3.2.6.1 DEFSYS (SYSTEM,NO.ARR,TRACE)

Define a new system. The system is inactive on completion of this command.

SYSTEM = Alphanumeric system identifier.

NO.ARR = Number of arrays in this system.

TRACE = Error trace (line number in calling routine).

C3.2.6.2 DELSYS (SYSTEM,TRACE)

Delete all references and storage associated with this system. The system is undefined on completion of this command.

SYSTEM = Alphanumeric system identifier.

TRACE = Error trace.

C3.2.6.3 ACTSYS (SYSTEM,TRACE)

Activate system for processing. A directory of associated arrays is opened and ready for processing. The system is active on completion of this command.

SYSTEM = Alphanumeric system identifier.

TRACE = Error trace.

C3.2.6.4 DEASYS (SYSTEM,TRACE)

Deactivate system from processing. The system is inactive on completion of this command and not available for processing unless reactivated by an ACTSYS command. The storage occupied by this system may be reallocated by the DBM (in which case the system is copied to auxiliary storage).

SYSTEM = Alphanumeric system identifier.

TRACE = Error trace.

C3.2.6.5 DEFARR (SYSTEM,ARRAY,NO.REC,NO.ROW,NO.COL,R.SIZE,TRACE)

Define a new array. The parent system must be active before any associated array can be defined. The array is inactive on completion of this command.

SYSTEM = Alphanumeric system identifier.

ARRAY = Alphanumeric array identifier.

NO.REC = Number of records used to store this array.

NO.ROW = Number of rows in array.

NO.COL = Number of columns in array.

R.SIZE = Record size (max.) used to store this array (default set by DBM using NO.REC).

TRACE = Error trace.

C3.2.6.6 DELARR (SYSTEM,ARRAY,TRACE)

Delete all references and storage associated with this array. The array is undefined on completion of this command. The storage occupied by array will be reallocated.

SYSTEM = Alphanumeric system identifier.

ARRAY = Alphanumeric array identifier.

TRACE = Error trace.

C3.2.6.7 ACTARR (SYSTEM,ARRAY,NO.REC,NO.ROW,NO.COL,R.SIZE,TRACE)

Activate array for processing. A directory of associated records is opened and ready for processing. The parent system must be active prior to array activation. The array is active on completion of this command.

SYSTEM = Alphanumeric system identifier.

ARRAY = Alphanumeric array identifier.

NO.REC* = Number of records used to store this array.

NO.ROW* = Number of rows in array.

NO.COL* = Number of columns in array.

R.SIZE* = Maximum record size associated with this array.

TRACE = Error trace.

*The arguments marked by * are returned by ACTARR.

C3.2.6.8 DEAARR (SYSTEM,ARRAY,TRACE)

Deactivate array from processing. The array is inactive on completion of this command and not available for processing unless reactivated by an ACTARR command. The storage occupied by the array may be reallocated by the DBM (in which case the array is copied to auxiliary storage).

SYSTEM = Alphanumeric system identifier.

ARRAY = Alphanumeric array identifier.

TRACE = Error trace.

C3.2.6.9 DEFREC (SYSTEM,ARRAY,RECORD,SIZE,F.COL,L.COL,E.BLK,TRACE)

Define new record to DBM. The parent array must be active before any associated record can be defined. The record is inactive on completion of this command.

SYSTEM = Alphanumeric system identifier.

ARRAY = Alphanumeric array identifier.

RECORD = Record number of record being defined.

SIZE = Size of this record. Must be less than R.SIZE used in defining parent array.

F.COL = First column to be stored in record.

L.COL = Last column to be stored in record.

E.BLK = Effecting block (used for equation solving only).

TRACE = Error trace.

C3.2.6.10 DELREC (SYSTEM,ARRAY,RECORD,TRACE)

Delete all references and storage associated with this record. The record is undefined on completion of this command.

SYSTEM = Alphanumeric system identifier.

ARRAY = Alphanumeric array identifier.

RECORD = Record number to be deleted.

TRACE = Error trace.

C3.2.6.11 ACTREC (SYSTEM,ARRAY,RECORD,F.COL,L.COL,E.BLK,TRACE)

Activate record for processing. The record data is brought into core, if necessary, ready for processing. The parent array must be active prior to activating the record.

SYSTEM = Alphanumeric system identifier.

ARRAY = Alphanumeric array identifier.

RECORD = Number of record to be activated.

F.COL* = First column stored in record.

L.COL* = Last column stored in record.

E.BLK* = Effecting block (equation solving only).

TRACE = Error trace.

*The arguments marked by * are returned by ACTREC.

C3.2.6.12 DEAREC (SYSTEM,ARRAY,RECORD,TRACE)

Deactivate record from processing. The record is inactive on completion of this command and not available for processing unless reactivated by any ACTREC command. The storage occupied by the record may be reallocated by the DBM (in which case the record is copied to auxiliary storage).

SYSTEM = Alphanumeric system identifier.

ARRAY = Alphanumeric array identifier.

RECORD = Record number being deactivated.

TRACE = Error trace.

C3.2.6.13 GETREC (DATA,SYSTEM,ARRAY,RECORD,F.COL,L.COL,E.BLK,TRACE)

Get a specified defined (but not necessarily active) record and copy it to a previously reserved area of core. If the specified record does not exist, the reserved area is initialized to zero. This command is designed to allow records to be copied to and from a buffer.

DATA - First word address of reserved area.

SYSTEM - Alphanumeric system identifier.

ARRAY - Alphanumeric array identifier.

RECORD - Record number to be copied.

F.COL* - First column of incoming record.

L.COL* - Last column of incoming record.

E.BLK* - Effecting block of incoming record.

TRACE - Error trace.

*The arguments marked by * are returned by GETREC.

C3.2.6.14 PUTREC (DATA,SYSTEM,ARRAY,RECORD,F.COL,L.COL,E.BLK,TRACE)

Put a specified record into DBM storage. If the record is not defined, PUTREC defines the record prior to transfer. This command is designed to allow initialization of records through the use of a buffer.

DATA - First word address of reserved area occupied by this record.

SYSTEM - Alphanumeric system identifier.

ARRAY - Alphanumeric array identifier.

RECORD - Record number to be transferred.

F.COL - First column stored in record.

L.COL - Last column stored in record.

E.BLK - Effecting block for this record.

TRACE - Error trace.

C3.2.6.15 LOCREC (SYSTEM,ARRAY,RECORD,TRACE)

Locate first word address of an active record.

LOCREC - First word address (function subprogram).

SYSTEM - Alphanumeric system identifier.

ARRAY - Alphanumeric array identifier.

RECORD - Record number to be located.

TRACE - Error trace.

C3.2.6.16 DEFFIL (SYSTEM,FILE,NO.REC,P.SIZE,P.FACT,TRACE)

Define a file. FILES are stored identically to arrays but require special access calls. The file is defined but inactive on completion of this command.

SYSTEM - Alphanumeric system identifier.

FILE - Alphanumeric file identifier.

NO.REC - Number of records initially assigned (can be automatically expanded by DBM).

P.SIZE - Packet size (size of data module to be manipulated).

PFACT = Number of packets to be stored per record.

TRACE = Error trace.

C3.2.6.17 DELFIL (SYSTEM,FILE,TRACE)

Delete all references and storage associated with this file. The storage area occupied by the file will be reallocated.

SYSTEM = Alphanumeric system identifier.

FILE = Alphanumeric file identifier.

TRACE = Error trace.

C3.2.6.18 ACTFIL (SYSTEM,FILE,SLOT,TRACE)

Activate file for processing. Reserve area in primary storage through which file records may be processed. On activation, a unique numerical identifier (SLOT) is assigned to each file. Only 20 files may be active at any time.

SYSTEM = Alphanumeric system identifier.

FILE = Alphanumeric file identifier.

SLOT = Slot number assigned to this file (returned by ACTFIL).

TRACE = Error trace.

C3.2.6.19 DEAFIL (SYSTEM,FILE,SLOT,TRACE)

Deactivate file by releasing the assigned SLOT. The storage occupied by this file may be reallocated by the DBM (in which case the file is copied to auxiliary storage).

SYSTEM = Alphanumeric system identifier.

FILE = Alphanumeric file identifier.

SLOT = File SLOT number.

TRACE = Error trace.

C3.2.6.20 GETFIL (DATA,SLOT,PACKET,TRACE)

Copy requested packet (PACKET) to reserved area (DATA) from file associated with slot number SLOT.

DATA = First word address of reserved area to which packet is to be copied.

SLOT = File SLOT number.

PACKET = Packet number to be copied.

TRACE = Error trace.

C3.2.6.21 PUTFIL (DATA,SLOT,TRACE)

Copy specified packet (PACKET) from reserved area (DATA) to file associated with identifier slot number SLOT. Each time a packet is updated it must be PUT to ensure that the DBM has the latest copy.

DATA = First word address of reserved area holding packet to be copied.

SLOT = File SLOT number.

TRACE = Error trace.

C3.2.6.22 ACTBUF (SLOT,SIZE,TRACE)

Reserve an area of size SIZE in core to be used as a buffer for processing. This buffer is assigned a unique numerical identifier SLOT.

SLOT - SLOT number assigned to this buffer (returned by ACTBUF).

SIZE - Buffer length.

TRACE - Error trace.

C3.2.6.23 DEABUF (SLOT,TRACE)

Deactivate buffer by deleting the reserved storage and releasing the assigned SLOT. The storage occupied by the buffer will be reallocated. The data in the buffer is *not* retained.

SLOT - SLOT number for buffer.

TRACE - Error trace.

C3.2.6.24 LOCBUF (SLOT,TRACE)

Locate first word address of buffer assigned to slot SLOT. Note that buffer addresses may be changed by other calls such as ACT*** and DEA***. Hence, all such calls should be performed prior to locating any first word addresses.

LOCBUF - First word address (function subprogram).

SLOT - SLOT number of buffer.

TRACE - Error trace.

C3.2.7 DBM ERROR MESSAGES

ERROR NO.	DESCRIPTION
101	Illegal system access packet number
102	System access table full
103	Undefined system
104	Inactive system
105	System array directory full
106	Undefined array
107	Inactive array
108	Illegal array access packet number
109	Array table full
110	Expansion of record size not currently available
111	Record size exceeds maximum or <0 for this field length
112	Illegal record access packet number
113	Record table full
114	Invalid priority value
115	Illegal core access packet number
116	Blank common exhausted
117	Unable to free file slot
118	Record undefined
119	Record inactive
120	Attempt to recover undefined record
121	Invalid slot number (<0 or >10)
122	Invalid block number (<0)
123	Illegal file name
124	Illegal buffer number
125	Inconsistent array name and buffer number
126	All buffer slots reserved.

C4. ADDITION OF ELEMENTS

SUMMARY

If a new element type is to be added to WIPS, new subroutines must be developed for the WIPS-MODL and WIPS-ANAL modules; modifications must be made to WIPS-RSLT and WIPS-EXEC; and a module to read in element property data must be developed. The chapters in this section describe the procedures to be followed.

CONTENTS

C4.1 INTRODUCTION

C4.1.1 GENERAL

If a new structural element is added to WIPS, new subroutines must be developed for the WIPS-ANAL, WIPS-MODL, and WIPS-RSLT modules; modifications must be made to WIPS-EXEC; and a module to read in the element property data must be developed.

WIPS-ANAL is the central analysis module of WIPS. It consists of a *base code* to which is added a library of element subroutine packages. WIPS-ANAL is designed to allow the addition of new elements with minimum knowledge of the base code. Each element subroutine package for WIPS-ANAL must contain certain subroutines with standard names and argument lists. These subroutines may call other subroutines if desired. Each standard subroutine performs well-defined processing tasks.

WIPS-ANAL uses a data management system (DMS) for all data storage and recall functions. The programmer is required to have some knowledge of the FILE management aspect of the DMS, but otherwise need not make use of the DMS. A description of the FILE management procedures is presented in Chapter C4.2. The particular FILES required for the element processing, and their contents, are described in Chapter C4.3. The required element subroutines, and the functions that they must perform, are outlined in Chapter C4.4.

WIPS-MODL and WIPS-RSLT also consist of base codes to which element subroutines must be added. However, whereas the element subroutines in WIPS-ANAL perform structural analysis calculations, the subroutines in WIPS-MODL are for interactive input of element data and preparation of data "cards" for WIPS-ANAL, and the subroutines in WIPS-RSLT are for output of element results. The procedures for adding new subroutines are described in Chapters C4.5 and C4.6, respectively.

WIPS-EXEC controls the execution and maintains the problem data base. The procedures for modifying WIPS-EXEC to add new file types and data input modules are described in Chapter C4.7. Notes on the coding of new data input modules are in Chapter C4.8.

C4.1.2 PRECISION

For WIPS-ANAL, all real variables are double precision, except as noted for subroutine OUTH**. For WIPS-MODL and WIPS-RSLT, all real variables are single precision. WIPS-EXEC does not use real variables.

C4.2 FILE MANAGEMENT

C4.2.1 FILE DEFINITION

A FILE in the DMS is a matrix of data consisting of a number of equal sized PACKETS. The FILE is stored automatically in the DATA BASE by the DMS. The data can be accessed only in single PACKET units. A PACKET can be any collection of data (mixed integer, alpha and real words). A PACKET is made available to any subroutine by moving it from the DATA BASE to a labelled common block. To develop new element subroutines, the programmer needs only to be concerned with the procedure for moving data PACKETS to and from the DATA BASE.

C4.2.2 MANIPULATION OF FILE PACKETS

Consider a labelled common block, /SAMPLE/, as follows:

COMMON /SAMPLE/ INTE(NI),FLOT(NF)

where NI,NF are constants defining the sizes of the FORTRAN arrays INTE and FLOT, containing integer and real variables, respectively. The content of /SAMPLE/ is a PACKET of data. A PACKET can be moved to and from a FILE (which resides in the DATA BASE) as follows:

(1) CALL GETFIL(INTE,KSAM,NPAC,ITAG)

This call to the DMS moves the PACKET from the DATA BASE to the /SAMPLE/ block.

(2) CALL PUTFIL(INTE,KSAM,NPAC,ITAG)

This call to the DMS moves the PACKET from the /SAMPLE/ block to the DATA BASE.

The arguments in the GETFIL and PUTFIL subroutines are as follows:

INTE = First word address to or from which the PACKET is to be passed (usually this will be the first word in the common block).

KSAM = FILE SLOT number. This SLOT number is an integer which is established when the FILE is activated some time prior to these calls. SLOT numbers required for element addition are automatically established in the WIPS-ANAL base code, and passed to the element subroutines, as explained subsequently.

NPAC = The number of the PACKET being moved. For example, if the FILE contains one packet for each element, NPAC is the element number.

ITAG = User-chosen integer number, which is unique to this subroutine, to help with error tracing. If the DMS detects an error, a message containing ITAG is printed.

C4.3 FILES REQUIRED FOR ELEMENT PROCESSING

C4.3.1 LIST OF FILES AND ASSOCIATED COMMON BLOCKS

The following FILES and associated labelled common blocks are used for element processing. When an element subroutine is called, certain PACKETS will already be resident in their respective blocks. These PACKETS are referred to as FIXED in the following list. The element programmer does not need to move data between these blocks and the data base. Other FILES are referred to as FREE. Data PACKETS in these FILES must be moved to and from their blocks, using the subroutine calls described in Chapter C4.2.

FILE Contents	PACKET Status	Labelled Common Block
1. Element Group Data	FIXED	/CEGC/
2. Element Control Data	FIXED	/CEIN/
3. General Element Data	FIXED	/CSHP/
4. Material Data	FREE	/CMAT/
5. Geometric Data	FREE	/CPRP/
6. Node Coordinates	FREE	/CXYZ/

C4.3.2 ELEMENT GROUP DATA (FIXED)

One element group data PACKET is set up for each element group. The PACKET contains the following:

```
COMMON /CEGC/ NGRP,IGRP,NELS,NFST,NDGR,MFER,
MFSR,NPSH,NEKU,IEGC(5),FEGC(4),NIEIN,NFEIN
```

where

NGRP = Element group number.

IGRP = Element type number.

NELS = Number of elements in group.

NFST = Element number of first element in group.

NDGR = Maximum number of nodes for any element in group (max. 21).

MFER = PACKET number minus 1, in Element Control Data FILE, of first element in group (i.e. PACKET numbers for elements are MFER+1, MFER+2, etc.).

MFSR = PACKET number minus 1, in General Element Data FILE, of first element in group.

NPSH = Number of PACKETS to be recalled from the General Element Data FILE for each element in group.

NEKU = Number of elements in group with tangent stiffness matrices which have changed since the last global stiffness reformulation.

IEGC(5) = Integer control data for element group, which varies with element type.

FEGC(4) = Real control data for element group, which varies with element type.

NIEIN = Number of integer items on each element data "card" (max. 6).

NFEIN = Number of real items on each element data "card" (max. 9).

This data is all preset by WIPS-ANAL and must not be modified in the element subroutine.

C4.3.3 ELEMENT CONTROL DATA (FIXED)

One element control data PACKET is set up for each element. The PACKET contains the following:

COMMON /CEIN/ IELN,NNOD,LCON(21),LUPD,IEIN(6),FEIN(9)

where

IELN = Element number.

NNOD = Number of nodes for element (generally equals NDGR).

LCON(I) = for I = 1, NNOD, contains coded data on the node number and degree of freedom numbers for element node I. The form of the data is described later.

LUPD = Element stiffness update indicator.

0 = element stiffness unchanged since last global stiffness reformulation;

1 = element stiffness changed.

IEIN(6) = Optional integer data for element, which varies with element type.

FEIN(9) = Optional real data for element, which varies with element type.

C4.3.4 GENERAL ELEMENT DATA (FIXED)

One general element data PACKET is set up for each element. The data contained in the packet varies with the element type. The PACKET must contain all data necessary to monitor the nonlinear response of this element. The general form is:

COMMON /CSHP/ ISHP(NI),FSHP(NF)

where

NI and NF = constants. NI must be rounded *up* to be an even integer (to allow for double precision conversion).

ISHP(NI) = alpha and/or integer data.

FSHP(NF) = real data.

C4.3.5 MATERIAL DATA (FREE)

One or more *material* data FILES is stored by the DMS. Any PACKET may be recalled, using a GETFIL call. A PACKET contains the following:

COMMON /CMAT/ EMAT(64)

where

EMAT(64) = *Material* data. This may be data of any type, but typically will be used to define the strength and stiffness properties of elements. The dimension of 64 is a maximum; frequently the actual length will be smaller.

C4.3.6 GEOMETRIC DATA (FREE)

One or more *geometric* data FILES is stored by the DMS. Any PACKET may be recalled using a GETFIL call. A PACKET contains the following:

COMMON /CPRP/ PROP(16)

where

PROP(16) = *Geometric* data. This may be data of any type, but typically will be used to define geometric properties for elements. The dimension of 16 is a maximum.

C4.3.7 NODE COORDINATE DATA (FREE)

FILES of node coordinates are stored by the DMS. Any PACKET may be recalled using a GETFIL call. A node coordinate data PACKET contains the following:

COMMON /CXYZ/ X,Y,Z

where, for PACKET number I,

X = X-coordinate for internal node number I

Y = Y-coordinate for internal node number I

Z = Z-coordinate for internal node number I

Internal node numbers are assigned by the base code, one for each external node.

C4.3.8 OTHER LABELLED BLOCKS

Three other labelled COMMON blocks may need to be used, as follows:

(1) Input-Output Block: COMMON /CIOC/ MDATA,MECHO,MSLOG,MIOC(9)

where

MDATA = Input unit number (e.g. READ(MDATA,1000)...) for reading data from the DATA file.

MECHO = Data echo unit number for echo-printing input data (the ECHO file).

MSLOG = Solution log unit number (the SLOG files).

MIOC(9) = Other units used by WIPS-ANAL.

The variables in this block must *not* be changed. MDATA and MECHO will typically be used for input and output. MSLOG is used for results envelopes and may be used for other special purposes (e.g. debugging). The other units should not be used.

(2) Scratch Data Block: COMMON /CWRK/ IWRK(512), FWRK(512)

where

IWRK(512) = any *temporary* alpha or integer data.

FWRK(512) = any *temporary* real data.

A programmer can use this block to hold temporary data in any subroutine during a single entry to that subroutine. The data is *not* preserved from one entry to the next.

(3) Command Block: COMMON /CMND/ ICMD,MALP(3),MINT(6),FLOT(5),JCMD(5)

where:

ICMD = Command word (e.g. 4HLOOP).

MALP(1) = Subcommand word within the main command (e.g. 4HMERR).

MALP(2) = Print key, to trigger diagnostic printing (4HPRIN or blank).

MALP(3) = Optional additional alpha key.

MINT(6) = Optional integer keys (e.g. number of cycles in LOOP).

FLOT(5) = Optional real keys (e.g. convergence tolerance).

JCMD(5) = Not currently used.

The data in this block must *not* be changed. It is unlikely that a programmer will need to make use of this block. The block needs to be referenced only if the computational algorithm of WIPS-ANAL is modified.

C4.4 ELEMENT SUBROUTINES FOR WIPS-ANAL

C4.4.1 SUBROUTINE NAMES AND FUNCTIONS

Thirteen subroutines which are called from the base code must be provided. The names of these subroutines and their functions are as follows:

SUBROUTINE NAME	FUNCTION
(1) ELHD**	Read element group data "card" and set up Element Group Data.
(2) ELRD**	Read element data "cards".
(3) ELIN**	Initialize General Element Data.
(4) STIF**	Compute change in element tangent stiffness.
(5) STAT**	Perform element state determination calculations.
(6) RINT**	Calculate internal resisting loads.
(7) DYLD**	Calculate initial dynamic loads for time step.
(8) FACT**	Calculate factor for next significant nonlinear "event".
(9) MERR**	Calculate midstep equilibrium error.
(10) ENVE**	Update element results envelope (maximum) values. Envelope values are typically not used by WIPS, so that this will usually be a dummy subroutine.
(11) OUTH**	Set up element results for writing by WIPS-ANAL to the RSLT file.
(12) EPRN**	Write envelope results (to SLOG file). Usually a dummy subroutine.
(13) PRIN**	Write element results to ECHO file (typically used for debugging only).

In each subroutine name, ** is a number which identifies the element type. For example, the names of subroutines for element type 5 must be:

ELHD05,ELRD05,ELIN05,STIF05,etc.

Any additional subroutines which are called by the above interface subroutines may be assigned any convenient names. It is suggested, however, that all have the same type number identifier in the subroutine name.

Explanations of the tasks to be performed by each of the subroutines, and descriptions of their argument lists, are given in the following sections. For illustration purposes, element type 5 is assumed.

C4.4.2 ELHD** - GROUP DATA INPUT

C4.4.2.1 Subroutine Statement

Subroutine ELHD** is called from the base code once for each element group. Its purpose is to initialize data in the /CEGC/ block and to write a group heading to the ECHO file. The subroutine statement must be of the form:

SUBROUTINE ELHD05

C4.4.2.2 Common Blocks

The /CEGC/ and /CIOC/ blocks must be included. The /CWRK/ block may be used but is not likely to be needed.

C4.4.2.3 Tasks to be Performed

The following tasks must be performed.

(1) Set NPSH in /CEGC/. The General Element Data is stored as NPSH packets of data, each consisting of 128 4-byte words.

(2) Set NDGR, NIEIN, and NFEIN in /CEGC/.

(3) Write, on unit MECHO, a heading identifying the element type. For example:

```
WRITE (MECHO,1000)
1000 FORMAT(5X, 14HU-BAR ELEMENTS)
```

C4.4.3 ELRD** - ELEMENT DATA INPUT

C4.4.3.1 Subroutine Statement

Subroutine ELRD** is called from the base code once for each element, or (if elements are generated) once for each generated set of elements. Its purpose is to read the element data "card" (or cards). The subroutine statement must be of the form:

SUBROUTINE ELRD05(JFEL,JLEL,JDIF,JNOD,JN,IEIN,REIN,IA)

where

JFEL = Number of element, or of first element in a generated set.

JLEL = Number of last element in generated set.

JDIF = Element number difference between successive elements in generated set.

JNOD = Node number difference between successive elements in generated set.

JN = Array into which NDGR node numbers are to be read (max. dimension = 21).

IEIN = Array into which optional integer data are to be read (max. dimension = 6).

FEIN = Array into which optional real data are to be read (max. dimension = 9).

IA = Not currently used.

C4.4.3.2 Common Blocks

The /CEGC/ block must be included. The /CWRK/ block may be used.

C4.4.3.3 Tasks to be Performed

The following tasks must be performed.

(a) Read one element card containing up to 16 integer data items in 16I5 format. The data consists of JFEL, JLEL, JDIF, and JNOD followed immediately by JN and IEIN. The total number of integer items to be read is: 4+NDGR+NIEIN. The first card contains the first 16 items.

(b) If JFEL = 0, this is the end of the element data. Return, without reading further cards.

(c) Read the balance of the integer items (if any), in 16I5 format.

(d) If NFEIN > 0, read NFEIN real data items, in 5E15.0 format.

C4.4.4 ELIN** - ELEMENT DATA INITIALIZATION

C4.4.4.1 Subroutine Statement

Subroutine ELIN** is called from the base code once for each element of the corresponding element type. Its purpose is to initialize the data in the /CEIN/ and /CSHP/ blocks. The subroutine statement must be of the form:

SUBROUTINE ELIN05(KXYZ,KMAT,KPRP)

where

KXYZ = Node coordinate SLOT number.

KMAT = Material data SLOT number.

KPRP = Geometric data SLOT number.

These slot numbers are assigned by the base code and must not be changed.

C4.4.4.2 Common Blocks

The /CEIN/ and /CSHP/ blocks must be included. The /CXYZ/, /CMAT/, and /CPRP/ blocks will generally, although not necessarily, be used. The /CIOC/ and /CWRK/ blocks may be included, if necessary.

C4.4.4.3 Data Available on Entry

The values of IELN, LCON, IEIN, and FEIN in /CEIN/ are set before entry. LCON contains the element node numbers at this stage.

C4.4.4.4 Tasks to be Performed

The following tasks must be performed.

(a) Set NNOD in /CEIN/.

(b) Set values for all variables in the /CSHP/ block, including zero initial values for all element forces, etc. The values in arrays LCON and IEIN will identify PACKET numbers which can be used to move packets to the /CXYZ/, /CMAT/, and /CPRP/ blocks (together with the SLOT numbers KXYZ, KMAT, and KPRP). Note, however, that only GETFIL calls should be used. Data should *not* be moved back using PUTFIL calls.

(c) Modify the LCON array to contain coded degree of freedom data for each node. The procedure is as follows.

C4.4.4.5 D.O.F. Code

The *D.O.F. code* for element node I is a six-word array, KDI(6), which identifies the translational and rotational displacements which are active degrees of freedom at node I. The degrees of freedom may be in a local element coordinate system or the global (substructure) coordinate system.

For example, the D.O.F. code for a 3D truss bar, in the global system, is (1,1,1,0,0,0). The sequence of terms is always the X, Y, Z, XX, YY, and ZZ displacements, respectively. In the D.O.F. code, 1 indicates that the displacement is an element degree of freedom, and 0 that the displacement has no effect on the element.

C4.4.4.6 Conversion of LCON

The FORTRAN function KODE converts an element node number and D.O.F. code to a single integer containing information on both. Modification of LCON(I) is typically performed with the following statements:

```
NODI    = LCON(I)
LCON(I) = KODE(KDI,NODI)
```

where

KODE = function name.

KDI(6) = D.O.F. code for element node I. This code will typically be the same for all nodes of an element, but may change from node to node, if desired. It must be set by the element programmer.

NODI = internal node number corresponding to element node I.

C4.4.5 STIF** - ELEMENT STIFFNESS CHANGE

C4.4.5.1 Purpose

Subroutines STIF** are called from the base code whenever a substructure stiffness matrix is to be modified. The purpose of the subroutine is to compute the *change* in element stiffness since the last entry to the subroutine. This subroutine is called only for elements with stiffnesses which have changed (as indicated by the variable LUPD in /CEIN/).

The subroutine statement must be of the form:

```
SUBROUTINE STIF05(IDUM,DUM(3))
```

where IDUM and DUM are not currently used.

C4.4.5.2 Common Blocks

The /CEIN/ and /CSHP/ blocks must be included. The /CEGC/ block may be needed. The /CWRK/ block will usually be used for the SE array (see below).

C4.4.5.3 Tasks to be Performed

The following tasks must be performed.

(a) Compute the change in element stiffness, and store in an array SE of appropriate size.

(b) Update any data in /CSHP/ which has changed. In general, this will be stiffness data and codes which indicate the element state.

(c) Call the stiffness assembly subroutine as follows:

```
CALL STFBLK(ITYP,IROT,1,NDOF,LCON,ROT,LE,SE)
```

where

ITYP = Stiffness storage scheme indicator for element, as follows:

- 0 = square symmetric;
- 1 = upper triangle, compacted column-wise;
- 2 = column compacted, irregular skyline profile.

IROT = Local-Global Indicator:

- 0 = stiffness formed in global coordinates;
- 1 = stiffness formed in local coordinates.

NDOF = Number of element degrees of freedom (number of rows in stiffness matrix).

LCON = Node D.O.F. and connectivity array, as before.

ROT = 3 x 3 local to global rotational transformation (required only if IROT = 1).

LE = Profile indicator array, giving locations of diagonal coefficients (required only if ITYP = 2).

SE = Element stiffness matrix.

C4.4.6 STAT** - STATE DETERMINATION

C4.4.6.1 Subroutine Statement

Subroutine STAT** is called for each element at frequent intervals during the analysis. Its purpose is to compute the element deformations (strains) and actions (stresses), and hence update the element state information. The subroutine statement must be of the form:

SUBROUTINE STAT05(D,V,A,DUM(3))

where

D,V,A = Increments in displacement, velocity, and acceleration at the element nodes, in the global directions. The arrays are passed to the subroutine in the form D(6,NNOD), V(6,NNOD), A(6,NNOD), where NNOD is the number of element nodes.

DUM(3) = Not currently used.

C4.4.6.2 Common Blocks

The /CSHP/ block must be included. The /CEGC/ and /CEIN/ blocks will usually be needed. The /CWRK/ and other blocks may be used.

C4.4.6.3 Tasks to be Performed

The specific state determination calculations to be performed vary with element type. In general, the element deformations are determined using the D array, and all element state quantities in the /CSHP/ block are then calculated and updated. The V and A arrays may be needed for some element types.

If the element changes such that its status is different from the status at the last stiffness update (because of material yielding, unloading, etc.), the stiffness update code, LUPD, in /CEIN/ must be set to 1. If large displacement effects are included for the element, LUPD will usually always be set to 1 (unless tolerances on change of shape are used).

C4.4.7 RINT** - INTERNAL RESISTING LOAD

C4.4.7.1 Subroutine Statement

Subroutine RINT** is called for each element at frequent intervals during the analysis. Its purpose is to compute the nodal loads which are in equilibrium with the current state of stress. The subroutine statement must be of the form:

SUBROUTINE RINT05(FE,FD,FM,IDUM)

where

FE,FD,FM = nodal elastic, viscous damping, and inertia forces to be returned by this subroutine. These are the external forces in equilibrium with the current state of stress in global coordinates. These forces must be transferred in the form FE(6,NNOD), FD(6,NNOD), FM(6,NNOD), where NNOD = number of element nodes.

IDUM = Currently not used.

C4.4.7.2 Common Blocks

The /CEIN/ and /CSHP/ blocks will usually be needed. Other blocks may be used.

C4.4.7.3 Tasks to be Performed

The current "elastic", damping (if any), and inertia (if any) nodal loads on the element must be calculated and returned in FE, FD, and FM, respectively. These loads are the external loads required to be *applied to the element* to satisfy element equilibrium.

C4.4.8 DYLD** - INITIAL DYNAMIC LOAD

C4.4.8.1 Subroutine Statement

Subroutine DYLD** is called for each element at the beginning of each time step. Its purpose is to compute the initial loads due to dynamic effects for the next time step. The subroutine statement must be of the form:

SUBROUTINE DYLD05(V,A,FI,DUM(4),IDUM)

where

V,A = Velocity and acceleration, as before.

FI = Initial load to be returned, in the form FI(6,NNOD).

DUM(4),IDUM = Not currently used.

C4.4.8.2 Common Blocks

The blocks /CEIN/ and /CSHP/ will generally be required. Other blocks may be used if needed.

C4.4.8.3 Tasks to be Performed

Calculate the contribution of this element to the initial load for the time step (the vector $\Delta \underline{R}_o$ in Section B1). For the existing WIPS elements, there are no contributions from internal element inertia, and damping contributions are present only when strain rate effects are considered. The precise procedure to be followed depends on the element characteristics. The existing DYLD** subroutines illustrate typical procedures.

C4.4.9 FACT** - EVENT FACTOR CALCULATION

C4.4.9.1 Subroutine Statement

Subroutine FACT** is called for each element at frequent intervals during the analysis. Its purpose is to calculate the proportion of a specified displacement increment which can be applied to an element before a significant nonlinear "event" occurs. Typical events are gap closure and inelastic unloading. The subroutine statement must be of the form:

SUBROUTINE FACT05(D,V,A,IDUM,EFAC,IVNT)

where

D,V,A = Displacement, velocity, and acceleration, as before.

IDUM = Not currently used.

EFAC = Event factor (the proportion of D required to produce the event, plus a small tolerance to ensure that the event is passed), to be returned. Return a value 1.0 if no event occurs.

IVNT(5) = Alpha description of event (5A4 - up to 20 characters), to be returned. Leave unchanged for no event. If this event governs, the description is written to the SLOG file to provide a record of the events.

C4.4.9.2 Common Blocks

The blocks /CEIN/ and /CSHP/ will generally be required. Other blocks may be used as needed.

C4.4.9.3 Tasks to be Performed

The following tasks must be performed.

(a) Calculate the event factor, EFAC. The decision on what type of nonlinearity to classify as an "event" must be made by the programmer. As a minimum, gap closure and inelastic unloading should be treated as events. Add a tolerance to the event factor to allow overshoot of the event. The tolerance should be chosen to allow only a small equilibrium error due to overshoot.

(b) If an event occurs, return a description (to be included in the SLOG file) of the event in IVNT(5).

C4.4.10 MERR** - MIDSTEP ERROR CALCULATION

C4.4.10.1 Subroutine Statement

Subroutine MERR** is called for each element at frequent intervals during the analysis. Its purpose is to calculate an approximation to the midstep equilibrium error, for use in time step selection by the base code. The subroutine statement must be of the form:

```
SUBROUTINE MERR (DV,DT,FM,IDUM)
```

where

DV = Increment of element nodal velocities (NNOD global values).

DT = Current time step.

FM = Midstep error to be returned (NNOD global forces).

IDUM = Not used.

C4.4.10.2 Common Blocks

The blocks /CEIN/ and /CSHP/ will usually be needed. Other blocks may be used

C4.4.10.3 Tasks to be Performed

The midstep error vector, FM, is to be calculated as

$$\underline{FM} = \underline{K}_t \cdot \underline{DV} \cdot DT/8$$

where $\underline{K}_t$ = element tangent stiffness.

C4.4.11 ENVE** - ENVELOPE UPDATE

C4.4.11.1 Subroutine Statement

Subroutine ENVE** is called for each element at the end of each time step. Its purpose is to accumulate envelope (peak) values of the element deformations (strains), actions (stresses), and any other element response quantities. This task is optional, depending on whether or not the programmer decides to store envelope values in the /CSHP/ block. The subroutine statement is as follows:

SUBROUTINE ENVE05(TIME)

where

TIME = Current time. The value is set by the base program.

C4.4.11.2 Common Blocks

The blocks /CEIN/ and /CSHP/ will usually be needed. Other blocks may be used.

C4.4.11.3 Tasks to be Performed

Envelope values will be stored in /CSHP/. The quantities to be enveloped must be chosen by the programmer. If the current value of any quantity exceeds the envelope value, the envelope value is replaced by the current value.

C4.4.12 OUTH** - TIME HISTORY SAVING FOR WIPS-RSLT

C4.4.12.1 Subroutine Statement

Subroutine OUTH** is called for each element at specified output intervals. Its purpose is to set up a selection of current element quantities to be saved on the RSLT file for subsequent post-processing by WIPS-RSLT. The subroutine statement must be of the form:

SUBROUTINE OUTH05(NVAL,VAL)

where

NVAL = number of values saved.

VAL = single precision array in which the values are placed.

C4.4.12.2 Common Blocks

The blocks /CEIN/ and /CSHP/ will be needed.

C4.4.12.3 Tasks to be Performed

Set the value of NVAL, and set up the selected quantities in array VAL.

C4.4.13 EPRN** - ENVELOPE VALUE PRINTOUT

C4.4.13.1 Subroutine Statement

Subroutine EPRN** is called for each element at specified envelope output intervals. Its purpose is to write the current envelope values on unit MSLOG (the SLOG file). The subroutine statement must be of the form:

SUBROUTINE EPRN05(TIME)

in which

TIME = Current time.

C4.4.13.2 Common Blocks

The blocks /CEIN/, /CSHP/, and /CIOC/ will be needed.

C4.4.13.3 Tasks to be Performed

Write the envelope values (formatted) on unit MSLOG. The data will appear at intervals on the SLOG file. Note that envelope values can also be obtained in WIPS-RSLT from the values saved on the RSLT file. The usual option in WIPS will be to omit envelope printout using the EPRN** subroutine.

C4.4.14 PRIN** - CURRENT STATE PRINTOUT

C4.4.14.1 Subroutine Statement

Subroutine PRIN** is called for each element at the specified output intervals. Its purpose is to write the current element results (status codes, stresses and strains, etc.) on the ECHO file. The subroutine statement must be of the form:

SUBROUTINE PRIN05(TIME)

where

TIME = Current time.

C4.4.14.2 Common Blocks

The blocks /CEIN/, /CSHP/, and /CIOC/ will be needed. Other blocks may be used.

C4.4.14.3 Tasks to be Performed

Write the data to be output (formatted) on unit MECHO. The data will appear in the ECHO file. The ECHO file will typically be used for debugging purposes only.

C4.5 ELEMENT SUBROUTINES FOR WIPS-MODL

C4.5.1 GENERAL

In the WIPS-MODL module, the analysis model is defined by means of a series of commands. Each command specifies data for either: (a) a single control point in a pipe run (a *zero length* command); or (b) some part of a pipe run between two nodes (a *nonzero length* command). The data in each command specifies: (a) the last control point in the region covered by the command; (b) the element or substructure type; and (c) a number of optional data items, which are different for each element type. Because of the differences in the optional data, the commands must be interpreted differently for each element type. This requires that a new interpretation subroutine be added to WIPS-MODL for each new element or substructure type. These subroutines are named OP****, where **** identifies the element or substructure type (e.g. OPPIPE, OPELBO).

The WIPS-MODL module also sets up formatted MODL files, which become data for the WIPS-ANAL module. The form of this data is described in Section C5. In the ELEMENT part of this data, "cards" specifying the elements must be set up. Because the required data varies for each element type, a new data card subroutine must be added to WIPS-MODL for each new element type. These subroutines are named DA**** (e.g. DAPIPE).

The requirements for these subroutines are described in this section.

C4.5.2 SUBROUTINE OP****

C4.5.2.1 Call Statement

A CALL statement must be added in SUBROUTINE KOMINP, following the calls to the existing OP**** subroutines. The call statement has exactly the same form as the existing statements, except that **** becomes the 4-character element type identifier.

C4.5.2.2 Subroutine Statement

The SUBROUTINE statement is as follows:

SUBROUTINE OP**** (IKDAT,RKDAT,IERR,IZL,IDFLT,ICPD,LICPD,CPD,
LCPD,NCPD,NCP1,NCP2,UNFAC,UNLFAC,UNFFAC)

The variables are as follows:

IKDAT (12) = Array for integer and/or alpha (a4) command data. See Section C5.6.

RKDAT (6) = Array for real command data. See Section C5.6.

IERR = Error code, to be set to 1 if data errors are detected.

IZL = Length code. If IZL = 0, this element must have zero length (e.g. "ubar" type); whereas if IZL = 1, the element must have nonzero length (e.g. "pipe" type). See later for method of use.

IDFLT = code for controlling the initialization of IKDAT and RKDAT, and for keying the print phase. See later for method of use.

ICPD (LICPD, NCPD) = Integer array from the COOR file. See Section C5 for array contents.

CPD (LCPD, NCPD) = Real array from the COOR file. See Section C5.

NCP1, NCP2 = Column numbers in the ICPD and CPD arrays of the control points from the immediately preceding command and the current command, respectively. These variables may be used to check for data errors (e.g. specifying straight elements in a curved pipe segment).

UNFAC(6) = Array of units scale factors, one for each item in RKDAT. See later for method of use.

UNLFAC = Units scale factor for length.

UNFFAC = Units scale factor for force.

Variables IKDAT, RKDAT, UNFAC, and (if an error is found) IERR must be set in the OP**** subroutine. The other variables are all set before entry, and must *not be changed.*

C4.5.2.3 The IOPT Array

In the subroutine, an array IOPT (4,NOPT) must be dimensioned. The value of NOPT is the number of optional data items which may be specified for this element type. The contents of each column of IOPT are as follows:

IOPT(1) = 4-character word (a4) denning option (e.g. 4HPROP for property set number; 4HBCON for boundary condition code).

IOPT(2) = 2-character word defining acceptable short form of option (e.g. 2HPR,2HBC).

IOPT(3) = data mode (1 = integer; 2 = real; -n = n-character alpha word, max = 4).

IOPT(4) = location in IKDAT or RKDAT were data is to be stored. Integer and alpha data must be placed in IKDAT, and real data in RKDAT.

For example, if IOPT(1) = 4HBCON, IOPT(3) = 1, and IOPT(4) = 2, the data following "BCON = " in the command is to be read as an integer and placed in IKDAT(2). See the existing OPBEAM and OPUBAR subroutines for further examples. The following restrictions *must* be observed:

(1) Column 1 of IOPT must be 4HPROP,2HPR, 1, 1.

(2) Column 2 of IOPT must be 4HBCON,2HBC, 1, 2.

Otherwise the optional data may be placed in any convenient locations in IKDAT and RKDAT.

C4.5.3 COMPUTATIONS IN OP****

C4.5.3.1 Typical Logic

Typical logic for the subroutine is shown in Table C4.1. The details can best be obtained by studying, for example, the existing OPBEAM and OPUBAR subroutines. The major requirements are described in the following sections.

C4.5.3.2 Set Units Factors

For each item in RKDAT, set the value of the units factors in UNFAC, using the factors UNLFAC and UNFFAC. For example, if the item in RKDAT(2) is a pressure, set UNFAC(2) = UNFFAC/UNLFAC**2.

C4.5.3.3 Print Data

If IDFLT = 2, then only a call to SUBROUTINE OPTPR is to be made. This subroutine sets up information on the optional data, for subsequent printing. The only reason for routing this call through OP**** is to gain access to the IOPT array. The CALL statement is as follows:

CALL OPTPR (IOPT, NOPT, IKDAT, RKDAT, UNFAC)

C4.5.3.4 Check Length

The variable IZL is set in the calling routine. If the control point in a command is the same as in the preceding command, then a single-node element type (e.g. a U-bar) must be specified, and IZL is set to zero. If the control point is not the same as in the preceding command, a finite length of pipe run is covered by the current command, a two-node element type (e.g. a pipe) must be specified. and IZL is set to 1. If IZL does not have an appropriate value for the specified element type, call SUBROUTINE WRONGL as follows:

CALL WRONGL (I, IERR)

where I = 1 for a two-node element type (e.g. in OPPIPE) and I = 2 for a one-node element type (e.g. in OPUBAR). Subroutine WRONGL prints an error message and sets IERR = 1.

C4.5.3.5 Set Default Values

In the calling routine, a check is made whether this is the first command in which this element type appears, or whether this element type has appeared in an earlier command. If the element type has appeared previously, the arrays IKDAT and RKDAT are initialized to the values in the *nearest preceding command* with that element type, and IDFLT is set to 1. If the element type has not been previously used, IKDAT and RKDAT are not initialized, and IDFLT = 0. The data is defaulted in this way so that it is not necessary to specify all optional data in each command, and less data needs to be input by the user.

In any OP**** subroutine, four types of optional data may be recognized, as follows.

(a) Type 1: The data must be specified in the current command (i.e. no default). This is rare.

(b) Type 2: The data must be specified, but can default to the preceding element of the same type (e.g. property set number).

(c) Type 3: The data may or may not be specified. If it is not specified it can default to the preceding element of the same type. If no such preceding element exists, it can default to a standard value (typically zero). An example of this is LOCL for a beam type element.

(d) Type 4: The data may or may not be specified. If it is not specified, it defaults to a standard value, regardless of whether or not a preceding element of the same type exists. An example of this is BCON. In the current version of WIPS-MODL, the BCON value defaults to zero (no constraint) at all nodes unless a specific value is input.

The value of IDFLT is used to control the initialization of IKDAT and RKDAT as follows:

(1) Always initialize Type 1 values to -1, and Type 4 values to the standard default.

(2) If IDFLT = 0, then in addition initialize Type 2 values to -1, and Type 3 values to the standard default.

When the optional data is extracted from the command, any specified data will override the default values, and any nonspecified data will leave the default values unchanged. The -1 values must be changed, otherwise some required data has been omitted (assuming -1 is not valid data: if so, initialize to some invalid value). This can be used to detect that required data is missing.

C4.5.3.6 Extract Command Data

Extract the optional data from the command, and store in IKDAT and RKDAT, by calling SUBROUTINE GETOPT. The CALL statement is as follows:

CALL GETOPT (IOPT, NOPT, IKDAT, RKDAT, IERR)

If data errors are detected (for example, real data in an integer field) an error message will be printed and IERR will be set to 1 in GETOPT, followed immediately by a return to OP****. In most cases, if IERR is 1 on return from GETOPT, OP**** should immediately return also. The user will be requested to re-enter the command.

C4.5.3.7 Check for Missing Data

Check for missing required data. If any Type 1 or Type 4 data is still initialized to -1, call SUBROUTINE MISOPT as follows:

CALL MISOPT (IOPT (1, I), IERR)

where IOPT(1, I) is the option name for the missing data. Subroutine MISOPT prints an error message and sets IERR = 1.

C4.5.3. Check Boundary Code

Check the boundary condition code. If IKDAT(2) is not zero, call CHKBC as follows:

CALL CHKBC (IKDAT (2), IERR)

Subroutine CHKBC checks that the code has the correct form (e.g. contains only ones and zeros). If the code is incorrect, an error message is printed and IERR is set to 1.

C4.5.3.9 Check Property Set

Check the property set number in IKDAT(1) by calling CHKSET as follows:

CALL CHKSET(IKDAT(1), IERR)

If the specified property set number exceeds the number of available property sets for this element type, an error message is printed and IERR is set to 1.

C4.5.3.10 Check Other Data

Perform other data checks as appropriate for the element type. It is important that no errors be permitted, to ensure that the data cards set up for WIPS-ANAL are error-free.

C4.5.3.11 Scale RKDAT

Multiply each item in RKDAT by the corresponding value in UNFAC.

C4.5.4 SUBROUTINE DA****

C4.5.4.1 Call Statement

A CALL statement must be added in SUBROUTINE DAELEM, following the calls to the existing DA**** subroutines. The call statement has exactly the same form as the existing statements, except that **** becomes the 4-character element type identifier.

C4.5.4.2 Subroutine Statement

The SUBROUTINE statement is as follows:

SUBROUTINE DA**** (NUME,NAMN,NESET,IUNIT,KOM,IKDAT,
RKDAT,XYZ,WGT,EMAT,LMAT,ICURV)

The variables are as follows:

NUME – Element number in element group. See WIPS-ANAL data structure for definition of an element group. In WIPS, all elements of a given type constitute a group.

NAMN(1) – Node number at element end i.

NAMN(2) – Node number at element end j (if element is a two-node type).

NESET – Material type number (the packet number in the EMAT array containing the properties for this element).

IUNIT – Unit number for DATA file. The data "card" for the element must be written on this file.

KOM(10) – KOM data for this element (see WIPS-MODL data structure).

IKDAT(12) – IKDAT data for this element (see WIPS-MODL data structure).

RKDAT(6) – RKDAT data for this element (see WIPS-MODL data structure).

XYZ(3,1) – X,Y,Z coordinates of node i.

XYZ(3,2) – X,Y,Z coordinates of node j, if used.

XYZ(3,3) – X,Y,Z coordinates of bend center, if this element lies on a bend in the pipe run.

WGT(3,1) =Element weight contributions at node i, in X,Y,Z directions (the same in all 3 directions for WIPS).

WGT(3,2) =Element weight contributions at node j, if used.

EMAT(LMAT) =EMAT data for this element (i.e. column NESET of the complete EMAT array).

All variables except WGT are set in the calling routine and must not be changed.

C4.5.4.3 Tasks to be Performed

The tasks to be performed are as follows:

(1) Set up the JN, IEIN, and REIN data for the element (see Section C4.4.3). Write the WIPS-ANAL data "cards" for this element on IUNIT.

(2) Calculate the weight (not mass) contributions of the element at nodes i and j, and return in array WGT.

The computations are typically simple. The existing DA**** subroutines illustrate the procedure.

C4.6 MISCELLANEOUS WIPS-MODL AND WIPS-RSLT MODIFICATIONS

C4.6.1 ELEMENT TYPE DATA

WIPS-MODL and WIPS-RSLT contain labelled COMMON blocks /ELTNAM/ and /OUTEL/, containing element type names and the number of output items for each element type. Provision is currently made for four element types (BEAM, UBAR, PIPE, and GAPF), plus the shell element (SHL4). Space is allocated for 15 more elements in arrays KEEEE(2,15) and IOOO(15).

If a new element type is added, make the following changes in BLOCK DATA subroutine ESBLK in WIPS-MODL and WIPS-RSLT.

(1) Insert a variable K****(2) into /ELTNAM/, where **** = element type name.

(2) Reduce the second dimension of KEEEE by 1.

(3) Initialize K**** to the element type name, using a DATA statement as for the existing element types.

(4) Increase NUMET by 1.

(5) Insert a variable IO**** into /OUTEL/, where **** = element type name.

(6) Reduce the dimension of IOOO by 1.

(7) Initialize IO**** to the number of output items for the element, using a DATA statement.

C4.6.2 OUTPUT UNITS

The force and length units must be specified on each entry to any WIPS module. Labelled COMMON block /UNITEL/ in BLOCK DATA subroutine ESBLK of WIPS-RSLT contains units conversion arrays for results processing.

Add an array IU****(2,NITEMS) to /UNITEL/ in ESBLK and subroutine TABELM, where **** = element type name and NITEMS = number of output items per element. This array contains units conversion data, as follows.

IU****(1,N) = length power in units for output item N.

IU****(2,N) = force power in units for output item N.

For example, if output item 3 is stress, then:

IU****(1,3) = -2

IU****(2,3) = 1

Initialize IU**** in ESBLK of WIPS-RSLT. In addition, at the end of TABELM, add a units conversion statement, following the same procedure as for the existing elements.

C4.7 WIPS-EXEC MODIFICATIONS

C4.7.1 PARAMETERS

WIPS-EXEC makes use of FORTRAN PARAMETER statements. Key parameters in WIPS-EXEC are represented by the following symbols:

GENERAL:

```
PARAMETER (ARB=1)
PARAMETER (MAINPG=10)
PARAMETER (MAXFIL=100)
PARAMETER (NO=0)
PARAMETER (YES=1)
```

CONTROL CHARACTERS:

```
PARAMETER (EOF=CHAR(4))
PARAMETER (EOL='')
PARAMETER (EOS='')
PARAMETER (NEWLIN=CHAR(10))
PARAMETER (TAB=CHAR(9))
```

CHARACTER STRING LENGTHS:

```
PARAMETER (ARGLEN=22)
PARAMETER (CMDLEN=21)
PARAMETER (INBFSZ=82)
PARAMETER (NUMLEN=10)
PARAMETER (PBNMSZ=21)
```

INPUT/OUTPUT:

```
PARAMETER (APPEND=6)
PARAMETER (BUFSIZ=21)
PARAMETER (DELFIL=1)
PARAMETER (KEEPFL=0)
PARAMETER (NEWFIL=4)
PARAMETER (OLDFIL=5)
PARAMETER (READMD=1)
PARAMETER (RECLEN=80)
PARAMETER (UNKNOW=9)
PARAMETER (WRITMD=2)
```

READ/WRITE LOGICAL UNIT NUMBERS:

```
PARAMETER (DELFD=15)
PARAMETER (NEWFFD=16)
```

```
PARAMETER (OLDFFD=17)
PARAMETER (STDERR=0)
PARAMETER (STDIN=5)
PARAMETER (STDOUT=6)
PARAMETER (WIPLFD=18)
PARAMETER (WIPSFD=19)
```

USER COMMANDS:

```
PARAMETER (UNKNCM=100)
PARAMETER (CATLCM=105)
PARAMETER (DELTCM=110)
PARAMETER (HELPCM=120)
PARAMETER (LISTCM=130)
PARAMETER (LISPCM=140)
PARAMETER (LISACM=150)
PARAMETER (LISCCM=155)
PARAMETER (QUITCM=160)
PARAMETER (PROBCM=170)
PARAMETER (ANALCM=200)
PARAMETER (BEAMCM=210)
PARAMETER (DATACM=215)
PARAMETER (ELBOCM=220)
PARAMETER (FRECCM=230)
PARAMETER (GAPFCM=240)
PARAMETER (MODLCM=250)
PARAMETER (MATLCM=255)
PARAMETER (PIPECM=260)
PARAMETER (GEOMCM=270)
PARAMETER (SLABCM=275)
PARAMETER (STRPCM=280)
PARAMETER (UBARCM=290)
PARAMETER (RSLTCM=300)
```

C4.7.2 COMMAND ADDITION

The procedure for adding commands (i.e. calls to new modules) to WIPS-EXEC is as follows.

1. Create a symbolic parameter "XXXXCM", where "XXXX" is the name of the command, and assign it a unique numeric command constant. For example, the "list" parameter is LISTCM with an associated constant of 130. The fortran PARAMETER statement takes the form:

   ```
   PARAMETER (LISTCM=130)
   ```

2. Insert the PARAMETER statement into the main program and function LOOKUP.

3. Create variable "SXXXX" in function LOOKUP, where "XXXX" represents the name of the command, and initialize it with a data statement.
4. Add the following code to the if-then-else statement in function LOOKUP:

   ELSE IF (EQSTR(COMAND,SXXXX) .EQ. YES) THEN

   LOOKUP = XXXXCM
5. Add the following code to the if-then-else statement in the main program:

   ELSE IF (CMD .EQ. XXXXCM) THEN

   CALL PROGRM("XXXX")
6. Update subroutine HELP to extend the list of commands.

C4.7.3 FILE TYPE ADDITION

See Section C5.1 for a description of the WIPS-EXEC file structure.

C4.8 DATA INPUT MODULES

C4.8.1 PURPOSE

Each data input module (e.g. WIPS-PIPE, WIPS-BEAM, etc.) accepts property set data interactively and stores each set as two records in a file of the same name (i.e. PIPE, BEAM, etc.). The first record must contain the following:

LEMAT: integer word, equal to length of EMAT array.

IENAM(10): character*4 array containing property set description.

The second record must contain the real array EMAT (LEMAT). This array contains all data for the property set. The maximum value of LEMAT is 64. Note that because EMAT is real, any integer property set items must be converted to real variables.

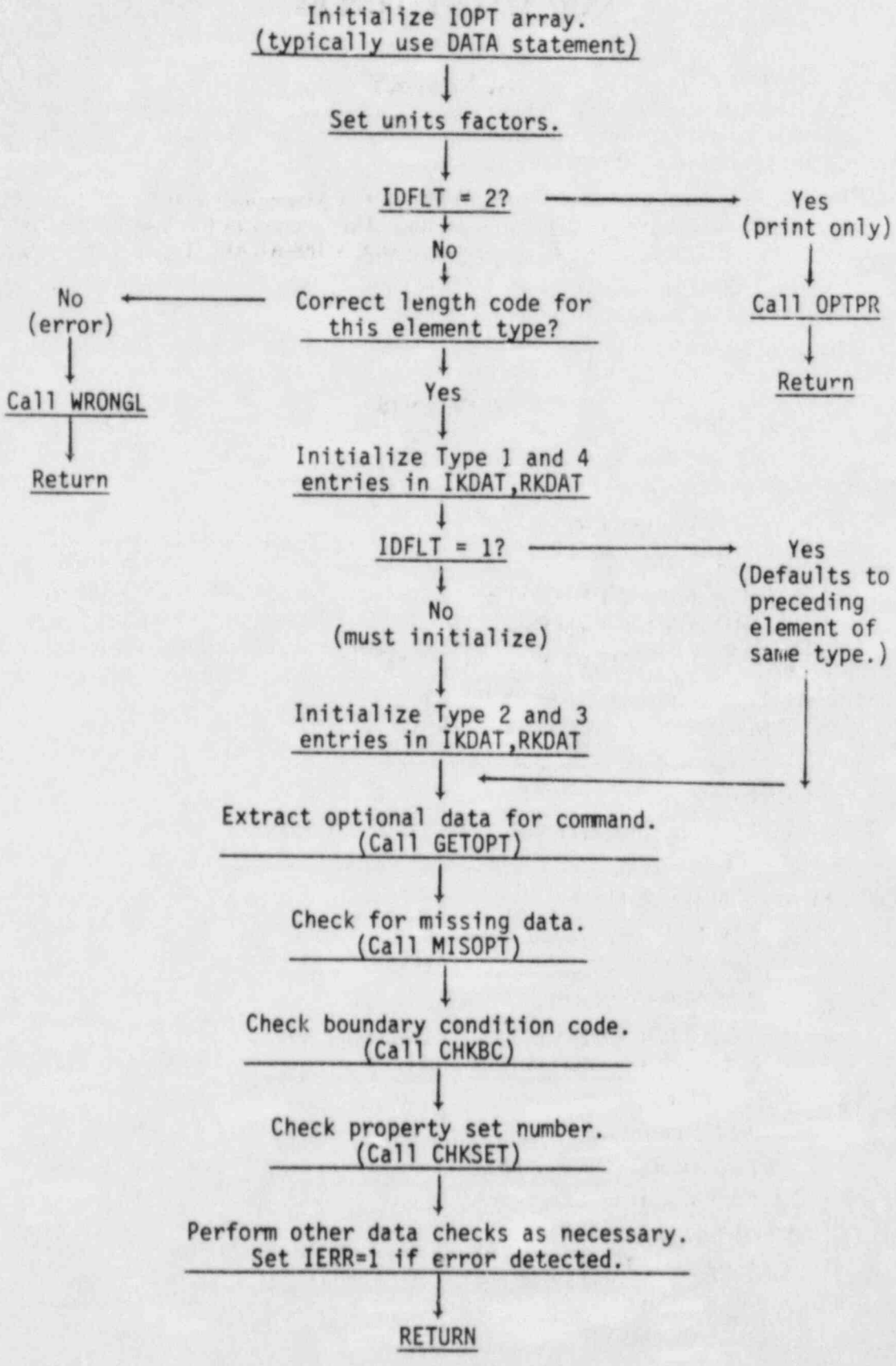
The data in EMAT varies with the element type. The EMAT contents are described for the currently available elements in Section C5.

C4.8.2 PROCEDURE

The detailed procedure varies with the element type. The module will typically make use of the freeform input package and may make use of the small data base manager. The procedure can best be learned by studying the existing modules.

It is important that the data input module perform exhaustive checks on the input data to ensure that the EMAT array contains correct and consistent information. The EMAT array is incorporated directly in the MODL file in the WIPS-MODL phase (and hence into the DATA file in the WIPS-DATA phase). Typically, no further consistency checks will be made in WIPS-MODL, so that errors must be detected in the input phase. If incorrect data is present in the DATA file, it may cause improper execution of WIPS-ANAL. Because of the complexity of WIPS-ANAL, data errors can be very difficult to locate.

TABLE C4.1 - FLOW DIAGRAM FOR OP**** SUBROUTINE



C5. FILE STRUCTURE

SUMMARY

In order to make modifications to the WIPS code, a programmer may require knowledge of the data contained in the WIPS-EXEC data base. The chapters in this section describe the data structures for all files except those associated with WIPS-ANAL. The data structure for WIPS-ANAL is described separately in Sections C6.

CONTENTS

C5.1.1 DATA STRUCTURE

C5.1.1.1 General

The WIPS-EXEC WIPSCAT file stores information about the data files produced by the WIPS computational modules. Each file is automatically cataloged by WIPS-EXEC, with its type, problem number, sequence number (if needed), date and time of creation, and an optional comment provided by the user. This information is grouped together in problem tables, several of which may be stored in the WIPSCAT file.

C5.1.1.2 WIPSCAT File Format

WIPSCAT is a direct-access unformatted file with a record length of 80 characters. Each problem table is preceded by a record which contains the table number and description, plus a pointer to the next table. The record layout is shown in Fig. C5.1.1. The size of each problem table is controlled by the MAXFIL parameter. Changing this parameter will not affect the use of existing problem tables.

C5.1.1.3 Problem Table Format

On entry to WIPS-EXEC, the first problem table in the WIPSCAT file is copied into memory to allow rapid access. Each record in the problem table contains the following variables:

Variable	Description	Length (Bytes)
FILENO	File Number	4
FTYPE	File Type	5
DATSTR	Date String	25
DESCRP	Comment about the File	36
	Unused	10

Unused records in the problem table are flagged by a zero file number.

C5.1.2 DATA STRUCTURE MANAGEMENT

The data modules are manipulated by the following routines:

WIPSCAT FILE:

FNDPRB - finds the position of a problem table in the WIPSCAT file corresponding to a particular problem number.

LODPRB - copies a problem table from the WIPSCAT file into core.

LSTCAT - prints information about the active files in all the problem tables held in the WIPSCAT file.

LSTPRB - prints the titles of all the problem tables held in the WIPSCAT file.

NAKPRB - makes a new problem table in the WIPSCAT file.

SAVPRB - copies the current problem table from core to the WIPSCAT file.

PROBLEM TABLE:

CATLOG - catalogs a file in the problem table.

DELETE - removes a file and its catalog entry from the problem table.

INPROB - locates a file name in the problem table.

INSTAL - puts information about a file into the problem table.

LIST - prints information about the active files in the current problem table.

LSTALL - prints information about all the files in the current problem table, including deleted files.

PRNTAB - copies the current problem table to the "oldfiles" file.

REPLAC - replaces file information in the problem table.

C5.1.3 STRING REPRESENTATION

Character strings are stored in character arrays and are terminated by a special character which is represented by the EDS symbolic constant. EDS translates into "char('')" when the program is compiled.

C5.1.4 SYSTEM DEPENDENT FEATURES C5.1.4.1 Subtask Execution

WIPS-EXEC executes the computational modules which constitute the WIPS system by issuing calls to the operating system via subroutine "execut". On the DEC VAX 11-780, subtasks may be spawned via a system call ("call system" in the UNIX operating system; "call sys$creprc" in the VMS operating system). If subtask spawning is not available, the computational modules may need to be run via an overlay mechanism.

C5.1.4.2 Date

Subroutine "fdate" fills a 24-character array with the date and time in the format shown by the following example. All the fields have constant width.

C5.1.4.3 Newline Suppression

The dollar sign ($) format control specifier suppresses the newline at the end of the last record of a formatted sequential write. It is typically used for terminal prompts.

C5.1.5 FILES

WIPSCAT - catalogs WIPS files.

WIPSLOG - records the input/output transactions from a WIPS interactive session.

newfiles - passes updated file information from a WIPS computational module to the WIPS-EXEC program.

oidfiles - passes file information from the WIPS-EXEC program to a WIPS computational module.

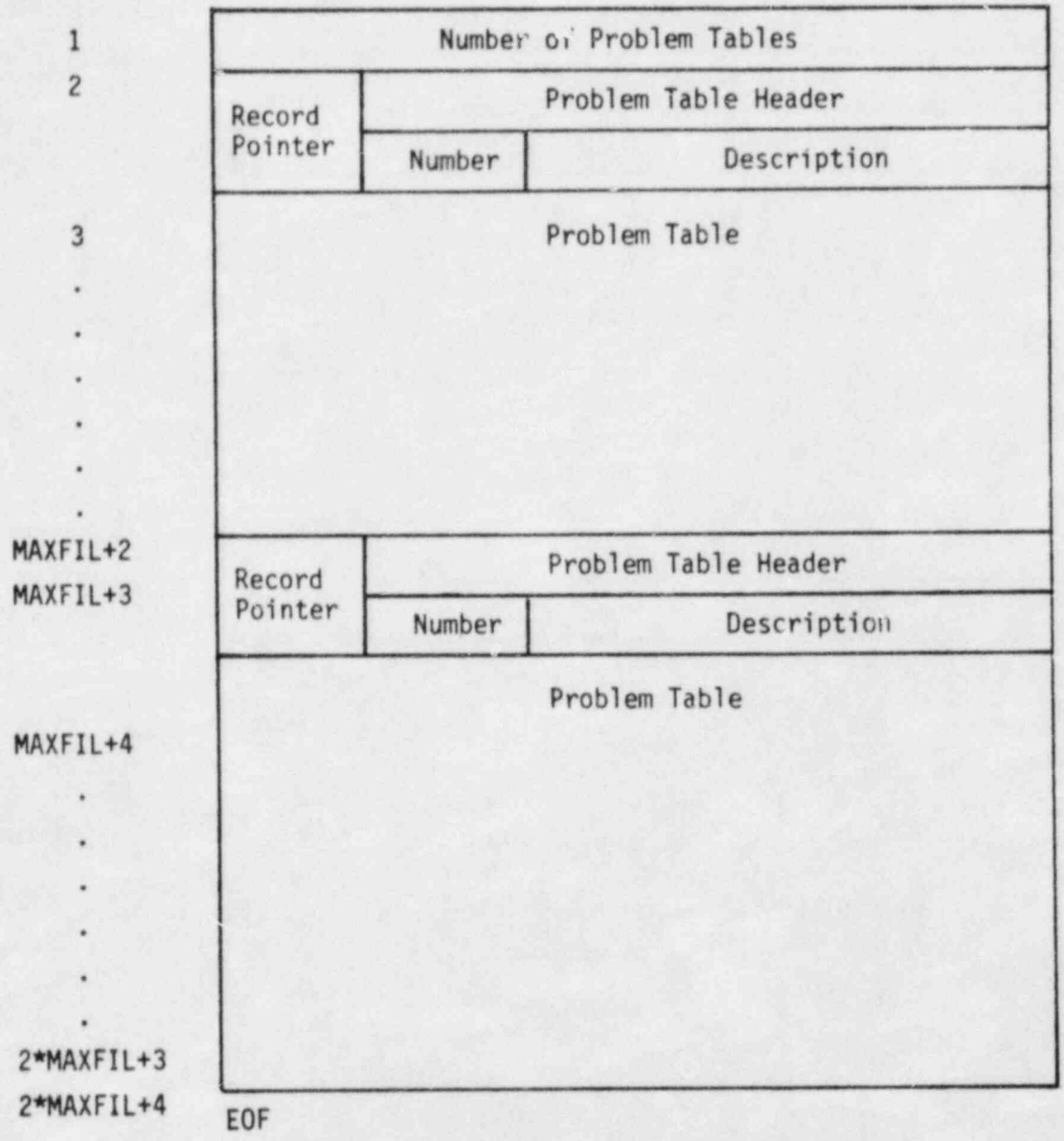


FIG. C5.1.1 - WIPSCAT FILE FORMAT

C5.2 WIPS-GEOM DATA FILES

C5.2.1 GENERAL

Data files of GEOM and COOR type are produced by the WIPS-GEOM module. A GEOM data file contains essentially the control point coordinate data as input, with only minor processing. A COOR file is obtained by processing a GEOM file and contains the global coordinates of all control points and extra nodes, plus other data. GEOM files can be recalled for editing. COOR files cannot be modified.

Both files are *unformatted.*

C5.2.2 GEOM FILE

C5.2.2.1 Control Record

A GEOM file consists of a control record plus two data records. The control record contains the following variables.

NCPS = number of control points

LICP = 8 = number of rows in integer data array ICP(LICP,NCPS)

LCP = 4 = number of rows in real data array CP(LCP,NCPS)

NRUNS = number of pipe runs

IPREC = precision of ICP array (1 = REAL*4; 2 = REAL*8)

C5.2.2.2 Integer Data Record

The first data record contains array ICP(LICP,NCPS). The column for each control point (c.p.) contains the following data:

ICP(1) = c.p. name (A4)

ICP(2) = c.p. type (A2) (DI, TI, TN, BR, or RF)

ICP(3) = coordinate option (A2) (DI, OF, TN, DU, ST, CU)

ICP(4) = number of extra nodes in preceding segment

ICP(5) = name of generation point i, if needed (A4)

ICP(6) = name of generation point j, if needed (A4)

ICP(7) = pipe run number

ICP(8) = uniqueness code for c.p. name (0 = unique; 1 = repeated elsewhere)

C5.2.2.3 Real Data Record

The second data record contains array CP(LCP,NCPS). The column for each c.p. contains the following data:

CP(1) = first coordinate data item, as input

CP(2) = second item

CP(3) = third item

CP(4) = bend radius (TI points only, otherwise zero)

C5.2.3 COOR FILE

C5.2.3.1 Control Record

A COOR file consists of a control record plus two data records. The control record contains the following variables.

NPTS = number of columns in ICPD, CPD arrays

LICPD = 13 = number of rows in integer data array ICPD(LICPD,NPTS)

LCPD = 5 = number of rows in real data array CPD(LCPD,NPTS)

NRUNS = number of pipe runs

IPREC = precision of CPD array (1 = REAL*4; 2 = REAL*8)

C5.2.3.2 Integer Data Record

The first data record contains array ICPD(LICPD,NPTS). The column for each node contains the following data.

ICPD(1) = c.p. name (A4); blank = extra node.

ICPD(2) = pipe run number.

ICPD(3) = c.p. type, as a number (1 = DI; 2 = TI; 3 = TN; 4 = BR; 6 = RF; 7 = center of curve (immediately follows TI); 0 = extra node).

ICPD(4) = node number

ICPD(5) = type of preceding element (1 = straight; 2 = curved).

The remaining entries in ICPD are used only for node coordinate generation in WIPS-GEOM.

C5.2.3.3 Real Data Record

The second data record contains array CPD(LCPD,NPTS). The column for each node contains the following data.

CPD(1) = X coordinate

CPD(2) = Y coordinate

CPD(3) = Z coordinate

CPD(4) = bend radius (TI points only)

CPD(5) = distance from beginning of pipe run.

C5.3 ELEMENT PROPERTY SETS

C5.3.1 FILE STRUCTURE

The data for each element property set consists of two records, as follows.

Record 1:

(1) LEMAT: Integer. Length of EMAT array. Max. 64.

(2) IENAM(10): Character*4. Property set description.

Record 2:

(1) EMAT(LEMAT): Real. Property set data.

All values are stored in kip and inch units.

C5.3.2 PIPE ELEMENT

LEMAT = 36. EMAT as follows.

1: Number of yield surfaces for material (min. 1, max. 2).
2: Poisson's ratio.
3-5: Material moduli for up to 3 segments.
6-7: Yield stresses.
8: Not used.
9: Radial error tolerance for material.
10: Angle tolerance for material.
11: Number of subelements in cross section.
12: Number of Gauss slices.. Usually 2.
13: Ovalling code (0=small; 1=large).
14: Number of ovalling hardening ratios (max. 2).
15: Scale factor for ovalling stiffness. Typically 1.0.
16: Scale factor for ovalling strength. Typically 1.0.
17: Outside diameter.
18: Wall thickness.
19: Self-weight per unit length.
20: Not used.
21-22: Ovalling hardening ratios.
23: Number of dashpot stiffnesses for material rate dependence (max. 3).
24-26: Dashpot stiffnesses.
27-29: Corresponding stress increases (last very large).
30: Strain rate tolerance for material. Not used.
31: Factor controlling ovalling geometric stiffness. Default value = 0.64.
32-36: Not used.

C5.3.3 BEAM ELEMENT

LEMAT = 64. EMAT as follows.

1: Yield surface type. Typically 1.

2: Angle tolerance for stiffness reformulation.

3-6: Yield surface exponents α_1 - α_4. Typically unused.

7: Overshoot tolerance for yield event (proportion of Y_1).

8: Reversal tolerance for unloading event (proportion of Y_1).

9-16: K_1, K_2, K_3, K_4, Y_1, Y_2, Y_3, GA' for M_{yy}.

17-24: K_1, K_2, K_3, K_4, Y_1, Y_2, Y_3, GA' for M_{zz}.

25-31: K_1, K_2, K_3, K_4, Y_1, Y_2, Y_3 for M_{xx}.

32: Not used.

33-39: K_1, K_2, K_3, K_4, Y_1, Y_2, Y_3 for F_x.

40: Strain rate tolerance.

41: Number of dashpot stiffnesses for rate dependence (max. 3).

42-44: Dimensionless rates for dashpot stiffness changes.

43-47: Dimensionless strength increases.

48: Weight per unit length.

49-64: Not used.

C5.3.4 UBAR ELEMENT

LEMAT = 64. EMAT as follows.

1: Number of segments in static force-extension curve (NSEG). Min. 2; Max. 6.

2: Number of segments in curve of force increase versus extension rate (NSEGV). Min. 0; Max. 3.

3: Default gap clearance.

4: Angle tolerance for stiffness reformulation.

5: Static stiffnesses (NSEG values).

5+NSEG: Static yield forces (NSEG values, last very large).

5+2*NSEG: Dashpot stiffnesses (NSEGV values).

5+2*NSEG+NSEGV: Dashpot "yield" forces (NSEGV values, last very large).

5+2*NSEG+2*NSEGV: Overshoot tolerance for gap closure and yield events (force units).

6+2*NSEG+2*NSEGV: Reversal tolerance for unloading event (force units).

7+2*NSEG+2*NSEGV: Separation tolerance for separation event (force units).

Rest not used.

C5.3.5 GAPF ELEMENT

LEMAT = 64. EMAT as follows.

1: Friction coefficient.

2: Tangent stiffness.

3: Not used.

4: Normal yield force (set very high).

5: Not used.

6: Normal stiffness.

7-8: Not used.

9: Angle tolerance for stiffness reformu'ation.

10: Gap closure tolerance (force units).

11: Not used.

12: Friction unloading tolerance (force units).

13: Friction yield (slip) tolerance (force units).

14: Gap opening tolerance (force units).

15-64: Not used.

C5.3.6 SHELL ELEMENT

LEMAT = 64. EMAT as follows.

1: Number of yield surfaces for material (min. 1, max. 4).

2: Number of dashpot stiffnesses for material rate dependence (min. 0, max. 3).

3: Radial error tolerance for material.

4: Angle tolerance for material.

5: Poisson ratio.

6: Weight density.

7: Strain rate tolerance.

8-9: Not used.

10-14: Static moduli (2 to 5 values).

15: Not used.

16-20: Static yield stresses (2 to 5 values; last must be very large).

21: Not used.

22-24: Material dashpot stiffnesses (0 to 3 values).

25-27: Stress increases at changes in dashpot stiffnesses (0 to values; last very large).

28-64: Not used.

C5.4 SUBSTRUCTURE PROPERTY SETS

C5.4.1 GENERAL

The data for each substructure property set varies with the substructure type. All values are stored in kip and inch units. All real variables are single precision.

C5.4.2 STRF SUBSTRUCTURE

Five records for each property set, as follows. Variables beginning with I through N are integer unless noted; all others are real.

Record 1: Control Record

(1) ISNAM(10): Character*4. Property set description.

(2) NREC: Number of following records (=4).

Record 2: Substructure Dimensions

(1) S1,SE,S2: Lengths of segments 1, 2, and 3 of substructure.

(2) D1,T1: Pipe diameter and wall thickness in segment 1.

(3) DE,TE: Pipe diameter and wall thickness in segment 2.

(4) D2,T2: Pipe diameter and wall thickness in segment 3.

(5) F1,F2: Mesh expansion factors in segments 1 and 2.

(6) ISNAM(10): Character*4. Property set description.

(7) N1,NE,N2: Numbers of longitudinal mesh divisions in segments 1, 2, and 3.

(8) NDIV: Number of circumferential mesh divisions.

(9) MTY(3): Material type identifiers for segments 1, 2, and 3 (currently 1, = standard Mroz).

(10) NMT(3): Material property set numbers (in MATL file) for segments 1, 2, and 3.

(11) DENS(3): Material weight densities for segments 1, 2, and 3.

(12) NGAUS(3): Numbers of Gauss points through thickness for segments 1, 2, and 3.

(13) NETY: Type number in WIPS-ANAL element library for shell element (=20).

Record 3: EMAT Array for Shell Elements in Segment 1

LEMAT = 36. EMAT as follows.

1: Number of yield surfaces for material (min. 1; max. 5).

2: Number of dashpot stiffnesses for strain rate effect (min. 0; max. 3).

3: Radial error tolerance for yield event (typically 0.02).

4: Angle tolerance for stiffness reformulation (typically 0.05 radians).

5: Poisson's ratio.

6: Weight density.

7-9: Not used.

10-15: Static moduli.

16-21: Yield stresses (last very large).

22-24: Dashpot stiffnesses.

25-27: Corresponding stress increases (last very large).

28-30: Coordinates (r,s,t) of output point 1 (0.,0.,1.).

31-33: Coordinates (r,s,t) of output point 2 (0.,0.,-1.).

34-36: Not used.

Record 4: EMAT Array for Shell Elements in Segment 2

As for Record 3.

Record 5: EMAT Array for Shell Elements in Segment 3

As for record 3.

C5.4.3 ELBO SUBSTRUCTURE

Five records for each property set, as follows. Variables beginning with I through N are integer unless noted; all others are real.

Record 1: Control Record

(1) ISNAM(10): Character*4. Property set description.

(2) NREC: Number of following records (=4).

Record 2: Substructure Dimensions

(1) S1,S2: Lengths of tangents 1 and 2.

(2) RAD,THETA: Bend radius and angle.

(3) D1,T1: Pipe diameter and wall thickness in tangent 1.

(4) DE,TE: Pipe diameter and wall thickness in elbow.

(5) D2,T2: Pipe diameter and wall thickness in tangent 2.

(6) F1,F2: Mesh expansion factors in tangents 1 and 2.

(7) ISNAM(10): Character*4. Property set description.

(8) N1,NE,N2: Numbers of longitudinal mesh divisions in tangent 1, elbow, and tangent 2.

(9) NDIV: Number of circumferential mesh divisions.

(10) MTY(3): Material type identifiers for tangent 1, elbow, and tangent 2 (currently 1, = standard Mroz).

(11) NMT(3): Material property set number (in MATL file) for tangent 1, elbow, and tangent 2.

(12) DENS(3): Material weight densities for tangent 1, elbow, and tangent 2.

(13) NGAUS(3): Numbers of Gauss points through thickness for tangent 1, elbow, and tangent 2.

(14) NETY: Type number in WIPS-ANAL element library for shell element (=20).

Record 3: EMAT Array for Shell Elements in Tangent 1

As for Record 3 in STRP substructure.

Record 4: EMAT Array for Shell Elements in Elbow

As for Record 3.

Record 5: EMAT Array for Shell Elements in Tangent 2

As for Record 3.

C5.4.4 SLAB SUBSTRUCTURE

Three records for each property set, as follows. Variables beginning with I through N are integer unless noted; all others are real.

Record 1: Control Record

(1) ISNAM(10): Character*4. Property set description.

(2) NREC: Number of following records (=2).

Record 2: Substructure Dimensions

(1) DIM(2): Lengths of OA, OB.

(2) PROP(3,2): Column 1 = first, center, and last strip widths as proportions of OA. Column 2 = same for OB.

(3) NDIV(3,2): Column 1 = number of divisions in first, center, and last strips along OA. Column 2 = same for OB.

(4) FEXP(3,2): Provision for mesh expansion factors. Not used.

(5) DCOSS(3,3): Row 1 = direction cosines of slab x axis with global X,Y,Z. Rows 2,3 = same for slab y,z axes.

(6) ORIG(3): Default coordinates of O.

(7) THICK: Slab thickness.

(8) DENS: Weight density.

(9) ISNAM(10): Character*4. Prope. ty set description.

(10) MTY: Material type (currently 1 = standard Mroz).

(11) NMT: Material property set number.

(12) NGAUS(2): Number of Gauss points through thickness in center region and outer regions.

(13) IBCOD(4): Character*4. Boundary condition types for OA, BC, OB, AC.

(14) NDIVT(2): Total subdivisions along OA,OB.

(15) NNODT: Total number of grid points.

(16) NETY: Type number in WIPS-ANAL element library for shell element (=20).

Record 3: EMAT Array for Shell Elements

As for Record 3 in STRP substructure.

C5.5 MATL AND FREC FILES

C5.5.1 MATL FILE

Two records for each property set, as follows. Variables beginning with I through N are integer unless noted; all others are real.

Record 1: Control Record

(1) LEMAT: Length of EMAT array (=36).

(2) MTYP: Character*4. Material type name. Currently "MROZ".

(3) MDESC(10): Character*4. Property set description.

Record 2: Property Data (EMAT Array)

Array EMAT(LEMAT) contains the following data.

1: Number of yield surfaces.

2: Number of strain rate segments.

3: Radial error tolerance (multiple of yield stress).

4: Angle tolerance for stiffness reformulation.

5: Poisson ratio.

6: Weight density.

7: Strain rate tolerance (multiple of strain rate).

8: Yield reversal tolerance (multiple of yield stress).

9: Not used.

10-15: Static moduli.

16-20: Yield stresses.

21: Not used.

22-24: Strain rate stiffnesses.

25-27: Corresponding strain rate limits (last very large).

28-36: Not used.

C5.5.2 FREC FILE

Two records for each record, as follows.

Record 1: Control Record

(1) NAMR: Character*4. Record name.

(2) NPAIR: Number of time-force pairs.

(3) ITITL(10): Character*4. Record description.

Record 2: Time-Force Values

Array RECV(2,NPAIR). First item in each column = time; second = force.

C5.6 WIPS-MODL DATA STRUCTURE

C5.6.1 COMMAND STRUCTURE

C5.6.1.1 General

WIPS-MODL accepts the definition of an analysis model as a series of *commands*. The commands are stored in a number of arrays which constitute the *command table*. When this table is complete, the commands are processed to produce a MODL file. The commands are extensively checked for consistency as they are input, to ensure that the MODL file contains no errors.

Consideration has been given to saving the command table (as a CMND file), to allow the table to be recalled and edited. This did not prove to be practicable, mainly because of the difficulty of checking an edited table for consistency. Hence, there is no provision for editing commands in the current version of WIPS. Certain features of the command table have, however, been chosen with future extensions of WIPS in mind to allow editing.

C5.6.1.2 Arrays

The command table consists of four arrays, namely a control array plus three arrays containing command data.

C5.6.1.3 Control Array

The control array consists of two words per command, as follows.

Word No.	Variable	Description
1	NKOM	Number of commands (columns) in command table.
2	KOM1	Column number of first command. This allows for a subsequent extension of WIPS-MODL to include editing of commands. Currently = 1.

C5.6.1.4 Command Data

The command data consists of three arrays, namely, KOM(10,NKOM), IKDAT(12,NKOM), and RKDAT(6,NKOM).

C5.6.1.5 KOM Array

Each column of KOM contains integer or alpha data for the corresponding command, as follows.

Row No.	Data
1	Line number. Lines are numbered sequentially in each segment.
2	Column number of next command.
3	Pipe run number.
4	Control point name (A4).
5	Element or substructure type name (A4). Blank = none.

6	Column number in ICPD and CPD arrays (see COOR file) for control point.
7	Element or substructure type number. Positive = element; negative = substructure; zero = none; 99 = zero element (e.g. 2 = UBAR; -2 = STRP).
8	Column number of preceding command.
9	Symmetry code (0 = full 3D; +1 = positive side of YZ plane; -1 = negative side of YZ plane; etc.).
10	Segment number.

C5.6.1.6 IKDAT Record

Each column of IKDAT contains integer or alpha data (optional command data, see Section A8), as follows.

Row No.	Data
1	Property set number for element or substructure.
2	Boundary condition code.
3	Name (A4) if a substructure.
4-8	Varies with element type. Determined by IOPT array in OP**** subroutine.
9	Longitudinal symmetry code (A4) (blank or "yes").
10	Not used.
11	Transverse symmetry code for STRP or ELBO (0 = none; 1 = use first half; 2 = use second half).
12	Longitudinal symmetry type (0 = none; 1 = YZ plane; 2 = ZX; 3 = XY).

C5.6.1.7 RKDAT Record

Each column of RKDAT contains real data (optional command data, see Section A8), as follows.

Row No.	Data
1	Lumped weight (weight units) at control point.
2-6	Varies with element type. Determined by IOPT array in OP**** subroutine.

C5.6.1.8 Units

All values are stored in kip and inch units. Unit conversions are made immediately after input and immediately before output.

C5.6.2 MODL FILE STRUCTURE

C5.6.2.1 WIPS-ANAL Input Data

The MODL file is formatted, mostly in 20A4 format. The file contains the same data as the DATA file (see Section C6) from the GBUILD command through the IMPACT data, with the following differences.

(1) The VELO data (if any) immediately follows the IMPACT data (or, if there is none, the MBUILD data).

(2) Following the VELO data is a LAST command. This separates the WIPS-ANAL data section of the MODL file from certain control information used by WIPS-DATA.

(3) Following the LAST command is control information for use by WIPS-DATA, as described in the following sections.

C5.6.2.2 WIPS-DATA Control Information

The control information consists of the following:

(1) One control record.

(2) A name-number comparison array (NODAT array).

(3) One set of records for each substructure.

(4) One record for each node in the main structure.

(5) One record for each output element.

(6) One record for each impact surface pair (if any).

C5.6.2.3 Control Record

The control record format is (8I5,I10). The items in the record are as follows.

(1) NNODAT: Number of rows in array NODAT.

(2) NMAINO: Number of main structure nodes.

(3) NITMN: Number of output items per main structure node (=9).

(4) NSUBO: Number of substructures for which results are output.

(5) MNSUB: Largest number of nodes in any substructure.

(6) NELMO: Total number of elements for which results are output.

(7) NIMPO: Number of impact surface pairs.

(8) NITIMP: Number of output items per impact surface pair (=6).

(9) LBUFO: Length of results output buffer (total number of output items per output time).

C5.6.2.4 NODAT Array

The NODAT array is written in (1X,A4,I5) format. The array contains, for each node in the main structure, the node number (from the COOR file) and the corresponding control point name. The COOR file node numbers are used to identify nodes in the DATA file. The NODAT array is used by WIPS-DATA to correlate c.p. names with node numbers for setting up DYLOAD data in the DATA file.

C5.6.2.5 Substructure Data

For each of the NSUBO output substructures, one control record is written plus one coordinate record for each substructure node. The control record is written in (2(1X,A4),I5,I10,8I5) format and contains the following items.

(1) Substructure name.

(2) Substructure type.

(3) Number of output nodes (currently all nodes).

(4) First word address in output buffer of results for first node.

(5) Number of output items per node (=3).

(6) Number of circumferential (or OA) subdivisions.

(7) Number of longitudinal subdivisions in segment 1 (or number of OB subdivisions).

(8) Number of subdivisions in segment 2.

(9) Number of subdivisions in segment 3.

(10) Total length of element output data (number of shell elements multiplied by 28).

(11) Transverse symmetry code (0 = 3D; 1 = first circumferential half; 2 = second half).

(12) Longitudinal symmetry code (0 = none; 1 = YZ plane; 2 = ZX plane; 3 = XY plane).

Each coordinate record is written in (3E15.8) format and contains the undeformed X,Y,Z coordinates of the corresponding node.

C5.6.2.6 Main Structure Node Data

One record is written for each main structure node in (1X,A4,I5,I10,I5,3E15.7) format. Each record contains the following items.

(1) Control point name.

(2) Node number (from COOR file).

(3) First word address in output buffer of results for this node.

(4) Pipe run number, set negative if node is at the beginning of a new model segment.

(5) Undeformed X coordinate.

(6) Y coordinate.

(7) Z coordinate.

C5.6.2.7 Element Data

One record is written for each output element in (2I5,I10,1X,A4,2I5) format. Each record contains the following items.

(1) Substructure number (0 = main structure).

(2) Element type number in WIPS-ANAL element library.

(3) First word address in output buffer of results for this element.

(4) Control point name at J end of element (main structure elements only).

(5) Circumferential (or OA) mesh location of element if in a substructure. Node number at J end if in main structure.

(6) Longitudinal (or OB) mesh location.

C5.6.2.8 Impact Data

One record is written for each impact surface pair in (1X,A4,I5) format. Each record contains the following items.

(1) Surface pair name.

(2) First word address in output buffer of results for this surface.

C5.6.2.9 First Word Addresses

Records containing first word addresses are written for use in setting up the THOUT data in the DATA file. One record is written for the main structure, plus one for each output substructure (if any), plus one for each impact surface pair (if any).

The main structure and substructure records are written in (2I10,4I5) format. Each record contains the following items.

(1) First word address in output buffer of results for first node.

(2) First word address of results for first element.

(3) Displacement/velocity/acceleration code. Set to 3 for main structure (all 3 quantities at each node) and 1 for substructures (displacements only).

(4) Rotation code. Set to 1 for both the main structure and substructures (rotations are not output).

(5-6) Not used.

The impact surface records are written in (I10,3I5) format. Each record contains the following items.

(1) First word address in output buffer of results for this surface pair.

(2) Results type code. Currently = 1 (6 output items per surface pair).

(3-4) Not used.

C5.7 WIPS-DATA OPERATIONS

C5.7.1 GENERAL

WIPS-DATA reads a MODL file, accepts additional data on loading and time step control, and produces a DATA file for WIPS-ANAL. In addition, WIPS-DATA initializes ECHO, SLOG, and RSLT files and creates empty PAUS and PAUZ files (if needed).

The DATA file is created by copying the MODL file (with some reorganization) and adding load and solution control data. The form of the DATA file is described in Section C6. The ECHO and SLOG files are merely created and initialized with a four-line description of the analysis. The RSLT file is initialized with the same analysis description plus a number of records which enable WIPS-RSLT to interpret the time-history records created by WIPS-ANAL.

C5.7.2 RSLT FILE

Following the four-line analysis description, the RSLT file is initialized with the following data.

(1) The control record described in Section C5.6.2.3, excluding NNODAT.

(2) The substructure data described in Section C5.6.2.5.

(3) The main structure data described in Section C5.6.2.6.

(4) The element data described in Section C5.6.2.7.

(5) The impact data described in Section C5.6.2.8.

(6) A record containing the word "LAST".

The NODAT array described in Section C5.6.2.4 is used by WIPS-DATA, then discarded. The first word address data described in Section C5.6.2.9 is used by WIPS-DATA to construct the THOUT command in the DATA file.

DISCLAIMER

This work was supported by the United States Nuclear Regulatory Commission under a Memorandum of Understanding with the United States Department of Energy.

Available from
GPO Sales Program
Division of Technical Information and Document Control
U.S. Nuclear Regulatory Commission
Washington, D.C. 20555

and

National Technical Information Service
Springfield, Virginia 22161

NRC FORM 335 (7.77)

U.S. NUCLEAR REGULATORY COMMISSION

BIBLIOGRAPHIC DATA SHEET

1. REPORT NUMBER (Assigned by DDC)
NUREG/CR-3686, Vol. 3
UCRL-15597

4. TITLE AND SUBTITLE (Add Volume No., if appropriate)
WIPS--Computer Code for Whip and Impact Analysis of Piping Systems - Part C- Programmer's Manual

2. (Leave blank)

3. RECIPIENT'S ACCESSION NO.

7. AUTHOR(S)
Graham H. Powell et al *

5. DATE REPORT COMPLETED
MONTH: March YEAR: 1983

9. PERFORMING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)
Lawrence Livermore National Laboratory
Post Office Box 808, L-46
Livermore, California 94550

*Subcontractor:
University of California
Berkeley, CA

DATE REPORT ISSUED
MONTH: June YEAR: 1984

6. (Leave blank)

8. (Leave blank)

12. SPONSORING ORGANIZATION NAME AND MAILING ADDRESS (Include Zip Code)
Division of Engineering Technology
Office of Nuclear Regulatory Research
U.S. Nuclear Regulatory Commission
Washington, D.C. 20555

10. PROJECT/TASK/WORK UNIT NO.

11. CONTRACT NO.
A0383-3

13. TYPE OF REPORT
Technical

PERIOD COVERED (Inclusive dates)

15. SUPPLEMENTARY NOTES

14. (Leave blank)

16. ABSTRACT (200 words or less)

WIPS (Whip and Impact of Piping Systems) is a special purpose computer code for the structural analysis of pipe whip dynamic effects following a postulated pipe rupture. WIPS has been developed primarily to provide support for the pipe whip analysis procedures described in Section 3.6.2 of the U.S. Nuclear Regulatory Commission Standard Review Plan.

This report summarizes the purpose and scope of the WIPS development effort, identifying those clauses in the standard Review Plan which refer to pipe whip analysis, and indicating how the WIPS code can be used to provide supporting data. Detailed information on use of the code is contained in accompanying reports which cover (1) use instructions, (2) theory, (3) programming procedures, and (4) verification examples.

17. KEY WORDS AND DOCUMENT ANALYSIS
pipe whip analysis
structural analysis

17a. DESCRIPTORS
WIPS Code

17b. IDENTIFIERS/OPEN-ENDED TERMS

18. AVAILABILITY STATEMENT
Unlimited

19. SECURITY CLASS (This report)
Unclassified

21. NO. OF PAGES

20. SECURITY CLASS (This page)

22. PRICE
$

NRC FORM 335 (7.77)

UNITED STATES
NUCLEAR REGULATORY COMMISSION
WASHINGTON, D.C. 20555