SANDIA REPORT
SAND86–0911 • Unlimited Release • UC–~~13~~
Printed September 1986

SUPES
A Software Utilities Package for the Engineering Sciences

Dennis P. Flanagan, William C. Mills-Curran, Lee M. Taylor

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-76DP00789

Printed in the United States of America
Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road
Springfield, VA 22161

NTIS price codes
Printed copy: A02
Microfiche copy: A01

SAND86-0911

Distribution
UC-13

Unlimited Release

Printed September 1986

SUPES
A Software Utilities Package for the Engineering Sciences

Dennis P. Flanagan
William C. Mills-Curran
Lee M. Taylor
Engineering Analysis Department
Sandia National Laboratories
Albuquerque, New Mexico 87185

SAND--86-0911

DE86 015749

ABSTRACT

The Software Utilities Package for the Engineering Sciences (SUPES) is a collection of FORTRAN subprograms which perform frequently used nonnumerical services for the engineering applications programmer. The three functional categories of SUPES are: (1) input command parsing, (2) dynamic memory management, and (3) system dependent utilities. The subprograms in categories one and two are written in standard FORTRAN-77, while the subprograms in category three are written to provide a standardized FORTRAN interface to several system dependent features.

[illegible]

ACKNOWLEDGMENT

The authors would like to acknowledge R. V. Lust of General Motors Research Laboratories who helped design the structure and logic of the memory manager.

CONTENTS

CONTENTS

CHAPTER 1

INTRODUCTION

The Software Utilities Package for the Engineering Sciences (SUPES) is a collection of FORTRAN subprograms which perform frequently used nonnumerical services for the engineering applications programmer. The three functional categories of SUPES are: (1) input command parsing, (2) dynamic memory management, and (3) system dependent utilities. The subprograms in categories one and two are written in standard FORTRAN-77, while the subprograms in category three are written to provide a standardized FORTRAN interface to several system dependent features.

Applications programmers face many similar user and system interface problems during code development. Because ANSI standard FORTRAN does not address many of these problems, each programmer solves these problems for his/her own code. SUPES aids the programmer by:

1. Providing a library of useful subprograms.

2. Defining a standard interface format for common utilities.

3. Providing a single point for debugging of common utilities. That is, SUPES has to be debugged only once and then is ready for use by any code.

Use of SUPES by the applications programmer can expand a code's capability, reduce errors, minimize support effort and reduce development time. Because SUPES was designed to be reliable and supportable, there are some features that are not included. (1) It is not extremely sophisticated, rather it is reliable and maintainable. (2) Except for the extension library (Chapter 4), it is not system dependent. (3) It does not take advantage of extended system capabilities, since they may not be available on a wide range of operating systems. (4) It is not written to maximize cpu speed.

Introduction

It is the intention of the authors to maintain SUPES on all scientific computer systems commonly used by Engineering Sciences Directorate (1500) staff. Versions of SUPES for new machines and/or operating systems will be added as needed. Other Sandia personnel may obtain copies of SUPES from the authors. SUPES will be available to non-Sandia personnel through the National Energy Software Center.

CHAPTER 2

FREE FIELD INPUT

This chapter describes the free field input system supported in SUPES. This software was developed because it was recognized that most codes written within the Engineering Sciences Directorate have very similar command input requirements. The SUPES free field input system consolidates the development and maintenance of command parsing code into a single set of reliable software. This utility provides a uniform command syntax across application codes to the end user, and minimizes the burden of command parsing on the applications programmer.

The design requirements which are imposed on the SUPES free field input system are as follows:

1) Input must follow a natural syntax which encourages readability.

2) The system must be applicable to both batch and interactive command input modes.

3) The software must be written in ANSI FORTRAN.

4) The interface to the applications program must be clear and flexible.

2.1 KEYWORD/VALUE INPUT SYSTEM

This section describes the basic characteristics of the SUPES free field input system. SUPES addresses the first two phases of command processing; it obtains a record from the input stream, and parses the record into logical components. Interpretation of the data in the final phase of command processing is left to the applications program.

SUPES provides a keyword/value input structure which encourages a verb oriented command language. The hallmark of this input style is the concept of "verbs" (or "keywords") which indicate how a command is to be interpreted. Since keywords allow each command to be self-contained, input lines need not follow a rigid order. This results in highly readable input data. For example, the command "YOUNGS MODULUS = 30.E6" has a very clear meaning. The verb oriented style can be contrasted with standard FORTRAN list-directed I/O which requires the application code to know precisely what to expect before reading a line of input.

The SUPES free field input system has a very simple, yet versatile syntax. Input records are broken into "fields". Each field is categorized according to its contents as: null, character, real, or integer. Note that these four categories form a hierarchy where each subsequent category is a more specific subset of the previous one. For example, "5.E3" is a real field because it can be interpreted as a REAL value as well as a valid CHARACTER string, but does not constitute a valid INTEGER format.

There are just three syntax markers in SUPES: field separators which delimit data fields, a comment indicator which allows a comment to be appended to command lines, and a continuation indicator which causes consecutive input records to be logically joined.

An application program need not heed all of the information returned for each field. A default value (blank or zero) is returned when a valid value is not specified for a given field. On the other hand, the application code can easily detect that the user has not explicitly specified a value so that a more meaningful default can be assumed, or so that the user can be prompted to supply more information.

2.2 SYNTAX RULES

The syntax rules for the SUPES free field input structure are listed below. This syntax describes how input records are parsed into data fields. Both the end user and the applications programmer should clearly understand these few rules.

1) A data field is any sequence of data characters within an input line. A data field is broken by (does not include) any nondata character or the end of the input line. A nondata character is a field separator, a space, a comment indicator, or a continuation indicator. Any other character is a data character.

2) A field separator is a comma (,), an equal sign (=), or a series of one or more spaces not adjacent to another separator.

3) A dollar sign ($) indicates a comment. All characters after and including the comment indicator are ignored.

4) An asterisk (*) indicates that the next input record will be treated as a continuation of the current line. All characters after and including the continuation indicator on the current line are ignored.

5) A null field does not contain any data characters. A null field can be defined explicitly only by a field separator (spaces cannot act as a field separator for an explicit null field). Fields which are not defined on the input line are implicitly null.

6) Lowercase letters are converted to uppercase. All other non-ANSI characters are converted to spaces.

7) A numeric field is a data field which adheres to an ANSI FORTRAN numeric format. A numeric field cannot be longer than 32 characters. A numeric field always defines a REAL (floating point) value; it also defines an INTEGER (fixed point) value if it adheres to a legal INTEGER format.

8) The maximum length of an input record is 132 characters.

Some important points which are not obvious from the above rules are noted below.

- Spaces have no significance, except when they act a field separator.

- Only the first occurrence of a comment or continuation character is significant; subsequent characters are considered part of the comment.

- A blank line has no data fields.

- If no data characters appear after the last field separator, the field after that separator will not be counted.

2.3 FREE FIELD INPUT ROUTINE (FREFLD)

The user interface to the SUPES free field input system consists of a single subroutine FREFLD. Input is prompted for, read, and echoed via this routine using specified I/O units. FREFLD returns the parsed data field values defined on the next input record and any continuation records. All I/O is accomplished via the utility routine GETINP, which is documented further in section 2.4.1.

The arguments to FREFLD are prescribed below.

```
      CALL FREFLD( KIN, KOUT, PROMPT, MFIELD, IOSTAT, NFIELD, KVALUE,
     *             CVALUE, IVALUE, RVALUE )
```

Argument:	KIN
Type:	INTEGER
Access:	Read Only
Description:	Unit from which to read input. If zero, read from the standard input device (terminal or batch deck) and echo to the standard output device (terminal or batch log). If nonzero, the caller is responsible for opening/ closing this unit.

Argument:	KOUT
Type:	INTEGER
Access:	Read Only
Description:	Unit to which to echo input. If zero, do not echo other than to the standard output device as described above. If nonzero, the caller is responsible for opening/ closing this unit.

Argument:	PROMPT
Type:	CHARACTER*(*)
Access:	Read Only
Description:	Prompt string. This string will be used to prompt for data from an interactive terminal and/or will be written as a prefix to the input line for echo. If the string 'AUTO' is specified, a prompt of the form ' n: ', where "n" is the current input line number (only lines read under the AUTO feature are counted), will be generated.

Argument:	MFIELD
Type:	INTEGER
Access:	Read Only
Description:	Maximum number of data fields to be returned. This value is the minimum permissible dimension of the output arrays described below.

Argument:	IOSTAT
Type:	INTEGER
Access:	Write Only
Description:	ANSI FORTRAN I/O status: IOSTAT < 0 - End of File IOSTAT = 0 - Normal IOSTAT > 0 - Error

Argument:	NFIELD
Type:	INTEGER
Access:	Write Only
Description:	Number of data fields found. If this value is less than MFIELD, the excess fields are implicitly defined as null fields. If this value is greater than MFIELD, the extra data fields are ignored.

Argument:	KVALUE
Type:	INTEGER Array
Access:	Write Only
Description:	Translation states of the data fields. The value of each element of this array is interpreted as follows: -1 = This is a null field. 0 = This is a nonnumeric field; only CVALUE contains a specified value. 1 = This is a REAL numeric field; CVALUE and RVALUE contain specified values. 2 = This is an INTEGER numeric field; CVALUE, RVALUE, and IVALUE contain specified values. The dimension of this array must be at least MFIELD.

Argument:	CVALUE
Type:	CHARACTER*(*) Array
Access:	Write Only
Description:	Character values of the data fields. The data will be left-justified and either blank-filled or truncated. The value in this array is set blank for a null field. The dimension of this array must be at least MFIELD. The character element size may be any value set by the caller.

Argument:	IVALUE
Type:	INTEGER Array
Access:	Write Only
Description:	Integer values of the data fields. The value in this array is set to zero for a null or non-INTEGER field. The dimension of this array must be at least MFIELD.

Argument:	RVALUE
Type:	REAL Array
Access:	Write Only
Description:	Floating-point values of the data fields. The value in this array is set to zero for a null or non-REAL field. The dimension of this array must be at least MFIELD.

2.3.1 Basic Examples

The following examples illustrate the operation of the SUPES free field input system.

```
INPUT RECORDS:
verb, 1 2. * continue on next line
key=5

RESULTS RETURNED FROM FREFLD:
 NFIELD =    5
    I      KVALUE(I)        CVALUE(I)              RVALUE(I)     IVALUE(I)
    1          0        "VERB               "      0.000E+00         0
    2          2        "1                  "       1.00             1
    3          1        "2.                 "       2.00             0
    4          0        "KEY                "      0.000E+00         0
    5          2        "5                  "       5.00             5
```

```
INPUT RECORD:
$ this is a comment line

RESULTS RETURNED FROM FREFLD:
 NFIELD =     0
      I      KVALUE(I)          CVALUE(I)              RVALUE(I)      IVALUE(I)
      1         -1        "                  "       0.000E+00          0
      2         -1        "                  "       0.000E+00          0
      3         -1        "                  "       0.000E+00          0
      4         -1        "                  "       0.000E+00          0
      5         -1        "                  "       0.000E+00          0


INPUT RECORD:
10,,

RESULTS RETURNED FROM FREFLD:
 NFIELD =     2
      I      KVALUE(I)          CVALUE(I)              RVALUE(I)      IVALUE(I)
      1          2        "10                "        10.0             10
      2         -1        "                  "       0.000E+00          0
      3         -1        "                  "       0.000E+00          0
      4         -1        "                  "       0.000E+00          0
      5         -1        "                  "       0.000E+00          0
```

2.4 UTILITY ROUTINES

The two routines described in this section, together with the FORTRAN extension library routines EXREAD and EXUPCS, are the only externals called by FREFLD. Application programs built on top of FREFLD may find further use for these routines.

2.4.1 Get Literal Input Line (GETINP)

All I/O for FREFLD is done through this subroutine. This routine was intentionally separated from FREFLD so that the caller can obtain an unmodified line of input (such as a problem title) via the same I/O stream. Applications which require a more complex syntax than SUPES provides (e.g., algebraic operations) may find GETINP advantageous.

There are four modes of operation of GETINP depending upon the specification of the I/O units KIN and KOUT. Each of these modes, which are summarized in the following table, may be useful to various applications.

KIN	KOUT	Source	Echo
0	0	Standard Input	Standard Output
0	M	Standard Input	Standard Output and File (M)
N	M	File (N)	File (M)
N	0	File (N)	none

The arguments to GETINP are prescribed below.

```
CALL GETINP( KIN, KOUT, PROMPT, LINE, IOSTAT )
```

Argument:	KIN
Type:	INTEGER
Access:	Read Only
Description:	Unit from which to read input. If zero, read from the standard input device (terminal or batch deck) and echo to the standard output device (terminal or batch log). If nonzero, the caller is responsible for opening/ closing this unit.

Argument:	KOUT
Type:	INTEGER
Access:	Read Only
Description:	Unit to which to echo input. If zero, do not echo other than to the standard output device as described above.

If nonzero, the caller is responsible for opening/closing this unit.

Argument:	PROMPT
Type:	CHARACTER*(*)
Access:	Read Only
Description:	Prompt string. This string will be used to prompt for data from an interactive terminal and/or will be written as a prefix to the input line for echo. If the string 'AUTO' is specified, a prompt of the form ' n: ', where "n" is the current input line number (only lines read under the AUTO feature are counted), will be generated.

Argument:	LINE
Type:	CHARACTER*(*)
Access:	Write Only
Description:	Line of input. This string will be blanked-filled or truncated, if necessary. The length of the string is set by the caller, but should not exceed 132.

Argument:	IOSTAT
Type:	INTEGER
Access:	Write Only
Description:	ANSI FORTRAN I/O status: IOSTAT < 0 - End of File IOSTAT = 0 - Normal IOSTAT > 0 - Error

2.4.2 Strip Leading/Trailing Blanks (STRIPB)

This routine is called by FREFLD from several locations. It may be useful to other applications as well. Note that STRIPB does not modify nor copy the input string, but simply returns the location of the first and last nonblank characters. If a substring is passed, these locations are relative the beginning of the substring. For example, if the substring STRING(N:) is passed to STRIPB, STRING(ILEFT+N-1:IRIGHT+N-1) would represent the result.

The arguments to STRIPB are prescribed below.

```
CALL STRIPB( STRING, ILEFT, IRIGHT )
```

Argument:	STRING
Type:	CHARACTER*(*)
Access:	Read Only
Description:	Any character string.

Argument:	ILEFT
Type:	INTEGER
Access:	Write Only
Description:	Relative index of the first nonblank character in STRING. ILEFT = LEN(STRING) + 1 if STRING = ' '.

Argument:	IRIGHT
Type:	INTEGER
Access:	Write Only
Description:	Relative index of the last nonblank character in STRING. IRIGHT = 0 if STRING = ' '.

CHAPTER 3

MEMORY MANAGER

The purpose of the memory manager utilities is to allow an applications programmer to write standard, readable FORTRAN-77 code while employing dynamic memory management for REAL, INTEGER, and LOGICAL type arrays.

Because the array sizes in most programs are problem dependent, the program's memory requirements are not known until the program is running. Since FORTRAN-77 does not provide for dynamic memory allocation, the programmer has to either predict the maximum memory requirement or use machine dependent requests for memory. In addition, dynamic memory allocation is an error prone exercise which tends to make the source code difficult to read and maintain.

The memory manager utilities are written in standard FORTRAN-77 and provide an interface which encourages readable coding and efficient use of memory resources. Machine dependencies are isolated through the use of the FORTRAN extension library (Chapter 4). All memory requests are in terms of numeric storage units (the amount of memory occupied by an integer, real, or logical datum [1]).

The memory manager utility is divided into three categories; basic routines, advanced routines, and development aids. These categories will be discussed in sections 3.2 through 3.4.

3.1 POINTER SYSTEM

In order to use the memory manager properly, the user must first understand the concept of a base array with pointers for accessing memory locations. The memory manager references all memory relative to a user supplied base array. A reference to memory is made in terms of an index or pointer to this base array. The pointers which the memory manager provides may take on a wide range of values, including negative numbers.

The base array must comply with the following rules:

1. The array must be of type INTEGER, REAL, or LOGICAL. Modified word length storage arrays such as INTEGER*2 or REAL*8 will result in invalid pointers with no error message.

2. The lower bound of the array subscript must be one.

The following FORTRAN statement defines a valid base array:

```
      DIMENSION A(1)
```

ONLY ONE BASE ARRAY MAY BE USED IN A PROGRAM.

In order to use memory allocated by the memory manager, the user merely needs to pass the base array with the correct pointer to a subprogram. For example, for a base array A and a pointer IP, a subroutine call would be:

```
      CALL SUBBIE ( A(IP) )
```

Although the programmer is not restricted to using the allocated memory in subprograms only, the recommended usage for the memory manager is to allocate dynamic arrays in the main program and then pass them to subroutines.

3.2 BASIC ROUTINES

The basic memory manager routines are those which are most commonly used and require little understanding of the internal workings of the utility.

3.2.1 Initialize (MDINIT)

The memory manager must be initialized with a call to MDINIT before any memory can be allocated. The main purpose of the initialization is to determine the location of the base array in memory.

```
CALL MDINIT (BASE)
```

Argument:	BASE
Type:	INTEGER, LOGICAL or REAL Array
Access:	Read Only
Description:	This array is used as a base reference to all dynamically allocated memory.

3.2.2 Define Dynamic Array (MDRSRV)

MDRSRV declares a new dynamic array. The user supplies the space required, and a pointer to the new space is returned. Note that the contents of the new storage are undefined.

```
CALL MDRSRV (NAME, NEWPNT, NEWLEN)
```

Argument:	NAME
Type:	CHARACTER*(*)
Access:	Read Only
Description:	This is the name of the new dynamic array. The memory manager will add this name to its internal dictionary; each array **must** have a unique name. The first eight characters are used for comparison, and leading and embedded blanks are significant.

Argument:	NEWPNT
Type:	INTEGER
Access:	Write Only
Description:	This is the pointer to storage allocated to this dynamic array relative to the base array.

Argument:	NEWLEN
Type:	INTEGER
Access:	Read Only
Description:	This is the length to be reserved for the new array. Any nonnegative number is acceptable. A zero length does not cause any storage to be allocated and returns a pointer equal to one.

3.2.3 Delete Dynamic Array (MDDEL)

MDDEL releases the memory that is allocated to a dynamic array.

```
CALL MDDEL (NAME)
```

Argument:	NAME
Type:	CHARACTER*(*)
Access:	Read Only
Description:	This is the name of the dynamic array which is to be deleted. The array name **must** match an existing name in the dictionary. The first eight characters are used for comparison, and leading and embedded blanks are significant.

3.2.4 Reserve Memory Block (MDGET)

MDGET reserves a contiguous block of memory without associating the block of memory with an array. MDGET should be called prior to a series of calls to

MDRSRV to improve efficiency and to reduce memory fragmentation. Further discussion of the operation of MDGET is found in section 6.2.1.

```
CALL MDGET (MNGET)
```

Argument:	MNGET
Type:	INTEGER
Access:	Read only
Description:	This specifies the desired contiguous block size.

3.2.5 Release Unallocated Memory (MDGIVE)

MDGIVE causes the memory manager to return unused storage to the operating system, if possible.

```
CALL MDGIVE ()
```

3.2.6 Obtain Statistics (MDSTAT)

MDSTAT returns memory manager statistics. MDSTAT provides the only method for error checking, and thus should be used after other calls to the memory manager to assure no errors have occurred.

```
CALL MDSTAT (MNERRS, MNUSED)
```

Argument:	MNERRS
Type:	INTEGER
Access:	Write Only
Description:	This is the total number of errors detected by the memory manager during the current execution.

Argument:	MNUSED
Type:	INTEGER
Access:	Write Only
Description:	This is the total number of words that are currently allocated to dynamic arrays.

3.2.7 Print Error Summary (MDEROR)

MDEROR prints a summary of all errors detected by the memory manager. The return status of the last memory manager routine called is also printed. MDEROR should be called any time an error is detected by a call to MDSTAT.

```
CALL MDEROR (IUNIT)
```

Argument:	IUNIT
Type:	INTEGER
Access:	Read Only
Description:	This is the unit number of the output device.

Error Codes

1	SUCCESSFUL COMPLETION
2	UNABLE TO GET REQUESTED STORAGE FROM SYSTEM
3	DATA MANAGER NOT INITIALIZED
4	DATA MANAGER WAS PREVIOUSLY INITIALIZED
5	NAME NOT FOUND IN DICTIONARY
6	NAME ALREADY EXISTS IN DICTIONARY
7	ILLEGAL LENGTH REQUEST
8	UNKNOWN DATA TYPE
9	DICTIONARY IS FULL
10	VOID TABLE IS FULL
11	MEMORY BLOCK TABLE IS FULL
12	OVERLAPPING VOIDS - INTERNAL ERROR
13	OVERLAPPING MEMORY BLOCKS - INTERNAL ERROR
14	INVALID MEMORY BLOCK - EXTENSION LIBRARY ERROR

3.2.8 Basic Example

```
DIMENSION BASE(1)
CALL MDINIT (BASE)
CALL MDGET (30)
CALL MDRSRV ('FIRST', I1, 10)
CALL MDRSRV ('SECOND', I2, 10)
CALL MDRSRV (' THIRD', I3, 10)
CALL MDSTAT (MNERRS, MNUSED)
IF (MNERRS .NE. 0) THEN
     CALL MDEROR (6)
     STOP
END IF
CALL MDDEL (' THIRD')
CALL MDGIVE ()
```

3.3 ADVANCED ROUTINES

The advanced routines are supplied to give added capability to the user who is interested in more sophisticated manipulation of memory. These routines are never necessary, but may be very desirable.

3.3.1 Rename Dynamic Array (MDNAME)

MDNAME renames a dynamic array from NAME1 to NAME2. The location of the array is not changed, nor is its length.

```
CALL MDNAME (NAME1, NAME2)
```

Argument:	NAME1
Type	CHARACTER*(*)
Access:	Read Only
Description:	This is the old name of the array. The first eight characters are used for comparison.

Argument:	NAME2
Type:	CHARACTER*(*)
Access:	Read Only
Description:	This is the new name of the array. The first eight characters are used.

3.3.2 Adjust Dynamic Array Length (MDLONG)

MDLONG changes the length of a dynamic array. The memory manager will relocate the array and move its data if storage cannot be extended at the array's current location. The user should assume that MDLONG **invalidates** the previous pointer to this array if the array is extended.

```
CALL MDLONG (NAME, NEWPNT, NEWLEN)
```

Argument:	NAME
Type:	CHARACTER*(*)
Access:	Read Only
Description:	This is the name of the dynamic array which the user wishes to extend or shorten.

Argument:	NEWPNT
Type:	INTEGER
Access:	Write Only
Description:	This is the new pointer to the dynamic array.

Argument:	NEWLEN
Type:	INTEGER
Access:	Read Only
Description:	This is the new length for the dynamic array.

3.3.3 Locate Dynamic Array (MDFIND)

MDFIND returns the pointer and length of storage allocated to a dynamic array. This routine would be used if the pointer from an earlier call to MDRSRV was not passed to a different subprogram.

```
CALL MDFIND (NAME, NEWPNT, NEWLEN)
```

Argument:	NAME
Type:	CHARACTER*(*)
Access:	Read Only
Description:	This is the name of the dynamic array to be located.

Argument:	NEWPNT
Type:	INTEGER
Access:	Write Only
Description:	This is the pointer to the dynamic array relative to the user's reference array.

Argument:	NEWLEN
Type:	INTEGER
Access:	Write Only
Description:	This is the length of the dynamic array.

3.3.4 Compress Storage (MDCOMP)

MDCOMP causes fragmented memory to be consolidated. Note that this may cause array storage locations to change. It is important to realize that all pointers **must** be recalculated by calling MDFIND after a compress operation. A call to MDCOMP prior to MDGIVE will result in the return of the maximum memory to the system.

```
CALL MDCOMP ()
```

3.4 DEVELOPMENT AIDS

The routines in this section are designed to aid the programmer during development of a program, and probably would not be used during execution of a mature program.

3.4.1 List Storage Tables (MDLIST)

MDLIST prints the contents of the memory manager's internal tables. Section 5.2.1 describes these tables.

```
CALL MDLIST (IUNIT)
```

Argument:	IUNIT
Type:	INTEGER
Access:	Read Only
Description:	This is the unit number of the output device.

3.4.2 Print Dynamic Array (MDPRNT)

MDPRNT prints the contents of an individual array.

```
CALL MDPRNT (NAME, IUNIT, NTYPE)
```

Argument:	NAME
Type:	CHARACTER*(*)
Access:	Read Only
Description:	This is the name of the array to be printed.

Argument:	IUNIT
Type:	INTEGER
Access:	Read Only
Description:	This is the unit number of the output device.

Argument:	NTYPE
Type:	CHARACTER*(*)
Access:	Read Only
Description:	NTYPE indicates the data type of the data to be printed; "R" for REAL, or "I" for INTEGER. Note that this is not necessarily the declared type of the base array.

CHAPTER 4

FORTRAN EXTENSION LIBRARY

The SUPES FORTRAN Extension Library provides a uniform interface to necessary operating system functions which are not included in the ANSI FORTRAN standard. This package makes it possible to maintain many codes on different operating systems with a single point of support for system dependencies. These routines provide very basic operating system support; they are not intended to implement clever features of a favorite system, to make FORTRAN behave like a more elegant language, nor to improve execution efficiency.

Each module included in the SUPES FORTRAN Extension Library must satisfy the following criteria:

1) The routine must provide a service which is beneficial to a wide range of users.

2) This task cannot be accomplished via standard FORTRAN.

3) This capability must be generic to scientific computers. Extension library routines must be supportable on virtually any system.

4) The routine must be codeable in FORTRAN so that the Extension Library can be implemented and maintained by FORTRAN programmers.

The SUPES FORTRAN Extension Library routines are designed to minimize the effort required to implement this software on a new operating system. Each interface is simple and straightforward. Operating system dependencies have been isolated at the lowest possible level.

4.1 USER INTERFACE ROUTINES

This section prescribes the calling sequence for FORTRAN Extension routines that are meant to be called directly from application programs.

4.1.1 Get Today's Date (EXDATE)

```
CALL EXDATE( STRING )
```

Argument:	STRING
Type:	CHARACTER*8
Access:	Write Only
Description:	Current date formatted as 'MM/DD/YY' where "MM", "DD", and "YY" are two digit integers representing the month, day, and year, respectively. For example, '07/04/86' would be returned on July 4, 1986.

4.1.2 Get Time of Day (EXTIME)

```
CALL EXTIME( STRING )
```

Argument:	STRING
Type:	CHARACTER*8
Access:	Write Only
Description:	Current time formatted as 'HH:MM:SS' where "HH", "MM", and "SS" are two digit integers representing the hour (00-24), minute, and second, respectively. For example, '16:30:00' would be returned at 4:30 PM.

4.1.3 Get Accumulated Processor Time (EXCPUS)

```
CALL EXCPUS( CPUSEC )
```

Argument:	CPUSEC
Type:	REAL
Access:	Write Only
Description:	Accumulated CPU time in seconds. The base time is undefined; only relative times are valid. This is an unweighted value which measures performance rather than cost.

4.1.4 Get Operating Environment Parameters (EXPARM)

```
CALL EXPARM( HARD,SOFT,MODE,KCSU,KNSU,IDAU )
```

Argument:	HARD
Type:	CHARACTER*8
Access:	Write Only
Description:	System Hardware ID. For example, 'CRAY-1/S'.

Argument:	SOFT
Type:	CHARACTER*8
Access:	Write Only
Description:	System Software ID. For example, 'COS 1.11'.

Argument:	MODE
Type:	INTEGER
Access:	Write Only
Description:	Job mode: 0 = batch, 1=interactive. For this purpose, an interactive environment means that the user can respond to unanticipated questions.

Argument:	KCSU
Type:	INTEGER
Access:	Write Only
Description:	Number of character storage units per base system unit.

Argument:	KNSU
Type:	INTEGER
Access:	Write Only
Description:	Number of numeric storage units per base system unit.

Argument:	IDAU
Type:	INTEGER
Access:	Write Only
Description:	Units of storage which define the size of unformatted direct access I/O records: 0 = character, 1 = numeric.

The ANSI FORTRAN standard defines a character storage unit as the amount of memory required to store one CHARACTER element. A numeric storage unit is the amount of memory required to store one INTEGER, LOGICAL, or REAL element. For this routine, a base system unit is defined as the smallest unit of memory which holds an integral number of both character and numeric storage units.

The last three parameters above can be used to calculate the proper value for the RECL specifier on the OPEN statement for a direct access I/O unit. For example, if NUM is the number of numeric values to be contained on a record and IDAU=0, set RECL = (NUM * KSCU + KNSU-1) / KNSU.

4.1.5 Get Unit File Name or Symbol Value (EXNAME)

```
CALL EXNAME( IUNIT,NAME,LN )
```

Argument:	IUNIT
Type:	INTEGER
Access:	Read Only
Description:	Unit number if IUNIT > 0, or symbol ID if IUNIT ≦ 0.

Argument:	NAME
Type:	CHARACTER*(*)
Access:	Write Only
Description:	File name or symbol value obtained from the operating system. It is assumed that the unit/file name or symbol/value linkage will be passed to this routine at program activation.

Argument:	LN
Type:	INTEGER
Access:	Write Only
Description:	Effective length of the string returned in NAME. Zero indicates that no name or value was available.

This routine provides a standard interface for establishing execution time unit/file connection on operating systems (such as CTSS) which do not support preconnection of FORTRAN I/O units. The returned string is used with the FILE specifier in an OPEN statement, as in the following example.

```
      CALL EXNAME( 10,NAME,LN )
      OPEN( 10,FILE=NAME(1:LN),... )
```

The symbol mode of this routine provides a standard path through which to pass messages at program activation. An example use is identifying the target graphics device for a code which supports multiple devices.

4.2 UTILITY SUPPORT ROUTINES

The routines prescribed in this section are intended primarily to support the SUPES free field input and memory manager utilities. While calling these routines directly will not disturb the internal operation of these other facilities, the use of EXMEMY (section 4.2.4) in conjunction with the memory manager is discouraged.

4.2.1 Convert String to Uppercase (EXUPCS)

```
CALL EXUPCS( STRING )
```

Argument:	STRING
Type:	CHARACTER*(*)
Access:	Read and Write
Description:	Character string for which lowercase letters will be translated to uppercase. All other characters which are not in the ANSI FORTRAN character set are converted to spaces.

4.2.2 Prompt/Read/Echo Input Record (EXREAD)

```
CALL EXREAD( PROMPT,INPUT,IOSTAT )
```

Argument:	PROMPT
Type:	CHARACTER*(*)
Access:	Read Only
Description:	Prompt string.

Argument:	INPUT
Type:	CHARACTER*(*)
Access:	Write Only
Description:	Input record from standard input device.

Argument:	IOSTAT
Type:	INTEGER
Access:	Write Only
Description:	ANSI FORTRAN I/O Status: IOSTAT < 0 - End of File IOSTAT = 0 - Normal IOSTAT > 0 - Error

This routine will prompt for input if the standard input device is interactive. In any case, the input line will be echoed to the standard output device with the prompt string as a prefix.

4.2.3 Evaluate Numeric Location (IXLNUM)

```
NUMLOC = IXLNUM( NUMVAR )
```

Argument:	NUMVAR
Type:	INTEGER or REAL
Access:	Read Only
Description:	Any numeric variable.

Argument:	NUMLOC
Type:	INTEGER
Access:	Write Only
Description:	Numeric location of NUMVAR. This value is an address measured in ANSI FORTRAN numeric storage units.

4.2.4 Get/Release Memory Block (EXMEMY)

```
CALL EXMEMY( MEMREQ,LOCBLK,MEMRTN )
```

Argument:	MEMREQ
Type:	INTEGER
Access:	Read Only
Description:	Number of numeric storage units to allocate if MEMREQ > 0, or release if MEMREQ < 0.

Argument:	LOCBLK
Type:	INTEGER
Access:	Read (release) or Write (allocate)
Description:	Numeric location of memory block. This value is an address measured in ANSI FORTRAN numeric storage units.

Only memory previously allocated to the caller via EXMEMY can be released via EXMEMY.

Argument:	MEMRTN
Type:	INTEGER
Access:	Write Only
Description:	Size of memory block returned in numeric storage units.

In allocate mode, MEMRTN < MEMREQ indicates that a sufficient amount of storage could not be obtained from the operating system. MEMRTN > MEMREQ indicates that the operating system rounded up the storage request.

In release mode, memory will always be released from the high end of the block downward. MEMRTN = 0 indicates that the entire block was returned to the operating system.

4.3 SKELETON LIBRARY

The Skeleton Library is an integral part of the SUPES Extension Library architecture. Each library module has a skeleton version which is written in fully standard FORTRAN. These routines are operational, but not fully functional. The skeleton routines serve as templates for implementing full support for the Extension Library on a new system. They also provide interim support during the development period so that the functional version of each module can be developed individually.

Application codes which call SUPES Extension Library routines should be structured to work with the Skeleton Library, albeit at a reduced level, whenever possible. This provides a consistent migration path for supporting these codes on a new system. The consequences of skeletal support for the Extension Library on higher level SUPES utilities is clearly documented in this report.

4.3.1 Skeleton Routine Specifications

The results produced by each Skeleton Library module are prescribed below.

1) EXDATE returns the string '00/00/00'.

2) EXTIME returns the string '00:00:00'.

3) EXCPUS returns zero.

4) EXPARM returns blank strings for hardware and software IDs, a zero which indicates batch mode, and unity for the three storage parameters.

5) EXNAME returns a null string; the result string is undefined and the length returned is zero.

6) EXUPCS converts all non-ANSI characters to spaces.

7) EXREAD simply reads from the standard input device.

8) IXLNUM returns unity.

9) EXMEMY allocates memory from the named COMMON block /EXTLIB/. The size of this static pool defaults to 1024, but can be changed by modifying a PARAMETER statement.

CHAPTER 5

SUPPORT PROGRAMMER'S GUIDE

This chapter documents the internal architecture for SUPES. It is intended to guide the maintenance of SUPES and support of SUPES on new operating systems. The consequences of using the Skeleton FORTRAN extension library on the internal operation of SUPES is fully discussed.

5.1 FREE FIELD INPUT

The SUPES free field input system consists of three subroutines: FREFLD (section 2.3), GETINP (section 2.4.1), and STRIPB (section 2.4.2). All of these routines are written in fully standard ANSI FORTRAN.

FREFLD calls the FORTRAN extension library routine EXUPCS (section 4.3.1). If only the skeleton version of EXUPCS is available, case insensitivity of input data (rule 6 of section 2.2) can not be guaranteed.

GETINP calls the FORTRAN extension library routine EXREAD (section 4.3.2). If only the skeleton version of EXREAD is available, GETINP will not prompt nor guarantee echo when reading from the standard input device (KIN = 0).

5.1.1 Implementation Notes on FREFLD

This section contains a basic outline of the internal operation of the free field input system and other supplemental information. More complete documentation is contained within the code itself.

FREFLD is organized into five phases:

1) All the output arrays are initialized to their default values.

2) The next input record is obtained via GETINP. Processing of a continuation line begins with this phase.

3) The effective portion of the input line is isolated by stripping any comment and leading/trailing blanks. A flag is set if a continuation line is to follow this record.

4) All field separators are made uniform. This phase streamlines the main processing loop which follows.

5) Successive fields are extracted, translated, and categorized until the input line is exhausted. After the maximum number of fields is reached, fields are counted but not processed further.

Upon leaving the main translation loop, the routine is restarted at phase 2 if the continuation flag is set.

The only errors returned by FREFLD are any returned from GETINP.

A data field is left-justified to define a CHARACTER value, but must be right-justified to obtain a numeric value. An internal READ is used to decode a numeric value from a data field. FREFLD relies upon the IOSTAT specifier to determine if the field represents a valid numeric format; this presents the possibility that some nonstandard numeric strings may be interpreted inconsistently by various operating systems. Default numeric values are overwritten if and only if IOSTAT indicates a valid translation.

CHARACTER data manipulation tends to be the area of lowest reliability for FORTRAN compilers, especially with supercomputers. An attempt was made in coding FREFLD to minimize the risk of triggering compiler bugs by manipulating pointers rather than shifting CHARACTER strings.

5.1.2 Test Program for FREFLD

A simple test program which calls FREFLD is included with the SUPES free field input system. FREFLD is instructed to digest data entered via the standard input device (e.g., keyboard), then the results are dumped to the standard output device (e.g., screen). This program should always be run to verify proper operation of FREFLD on a new operating system or compiler. Application programmers are encouraged to experiment with this program to learn what to expect from FREFLD.

5.2 MEMORY MANAGER

This section includes details of the internal operations of the memory manager, assumptions used in the memory manager, and details on the implementation of the memory manager on systems which do not support the extension library.

5.2.1 Table Architecture and Maintenance

The bookkeeping for the memory manager is accomplished with three tables; a memory block table, a void area table, and a dictionary.

The memory block table maintains a record of contiguous blocks of memory that have been received from the operating system. If a series of requests causes separate blocks to become contiguous, these blocks are joined. The beginning location and length of each memory block is recorded, and the table is sorted in location order.

Within each memory block, sections of memory that are not currently allocated to arrays are recorded in the void area table. As in the case of the memory block table, contiguous voids are joined and this table is sorted in location order.

The dictionary relates storage locations with eight character array names. The dictionary is sorted via the default FORTRAN collating sequence. All characters (including blanks) are significant. All names are blank filled or truncated to eight characters. In addition to the array name, the dictionary stores the location and length of each dynamic array.

Any call for memory (MDGET or MDRSRV) will be satisfied in one of two ways:

1. If a void of sufficient size is available, then this void will be used for the new array (MDRSRV). In the case of MDGET, no further action is taken.

2. An extension library call (EXMEMY) is made to get more memory from the system.

A request to extend an array (MDLONG) is satisfied in one of three ways:

1. If a void of sufficient size exists at the end of the array, then this space is allocated to the array.

2. If a void large enough for the extended array exists elsewhere in memory, the array is moved to this location. Note that the data is actually shifted and the pointer is updated.

3. An extension library call (EXMEMY) is made to get more memory from the system.

A call to MDCOMP will cause all arrays within each memory block to be moved to the lower addresses (pointers) within that memory block. Thus, all voids in the block will be joined at the end of the block.

A call to MDGIVE will attempt to return memory to the system. Only voids at the end of a memory block are subject to this attempt, and the system may accept only portions of these. Thus a call to MDCOMP followed by MDGIVE will release the maximum memory to the system.

5.2.2 Non-ANSI FORTRAN Assumptions

Although the memory manager is written in standard FORTRAN-77, it does depend on some assumptions which are not part of the ANSI standard. These assumptions are:

1. The contents of a word are not checked nor altered by an INTEGER assignment. Data is moved by MDLONG or MDCOMP as INTEGER variables.

2. Strong typing is not enforced between dummy and actual arguments. This allows the same base array to pass storage to any INTEGER, REAL, or LOGICAL array.

3. Array bounds are not enforced. Thus, any value is a valid subscript for the base array.

4. All dynamically allocated memory must remain fixed in relation to the base array.

5.2.3 Standard FORTRAN Implementation

If an installation does not yet support the extension library, it is still possible and advantageous to use the memory manager. In this case, the memory manager will act as a dynamic allocator of static (already dimensioned) memory. Codes which employ the memory manager therefore do not need to be rewritten, and codes under development can anticipate the implementation of the extension library.

When the subprograms IXLNUM or EXMEMY of the extension library are not available, the following steps must be taken before using the memory manager:

1. Install the skeleton version of the extension library (Section 4.3.1).

2. Alter the memory manager subroutine MDINIT as follows:

   <u>ORIGINAL</u>

   ```
   DIMENSION MYV(1)
   ```

   <u>ALTERED</u>

   ```
   PARAMETER (MAXSIZ=1024)
   COMMON /EXTLIB/ MYV( MAXSIZ)
   ```

3. Put the base vector in the user's program in the COMMON block EXTLIB and dimension it consistently with the COMMON blocks in EXMEMY and MDINIT.

4. If more than 1024 numeric storage units are required, change the parameter statement in MDINIT, EXMEMY and the user's program.

5.2.4 Test Program

In order to aid the installation of the memory manager at a new site, an interactive test program has been written which allows the user to exercise each of the features of the memory manager and insure that it is operating properly.

5.3 FORTRAN EXTENSION LIBRARY IMPLEMENTATION

Implementing the SUPES FORTRAN extension library on a new operating system requires a firm understanding of that system, but should not require a great deal of programming. Since the package is by definition system dependent, it is impossible to predict the exact procedure which will be required to implement these routines on a given operating system. This section provides some general guidelines and hints compiled from experience in implementing the package on several very different systems.

The FORTRAN extension library routines should be coded in FORTRAN whenever possible so that the package can be maintained on a given system in the absence of the original implementor. The code should be extensively commented and references to appropriate system manuals should be included.

It is generally best to start with the skeleton library routines and gradually add system dependent code to provide full capability. Concepts should be drawn from extension library versions from other systems before outlining a plan of attack for the new system.

It is suggested that extension library modules be implemented in the following order:

1) EXUPCS. The skeleton version should be sufficient.

2) EXTIME, EXDATE, EXCPUS, IXLNUM, and EXPARM. These routines are generally straightforward and can be accomplished simply with the aid of the FORTRAN manual for the particular operating system.

3) EXREAD, EXNAME, and EXMEMY. These routines require a more intimate knowledge of the operating system. A substantial set of system documentation may be required to accomplish these tasks.

5.3.1 Implementation Notes for Modules

The format of the date for EXDATE must be strictly observed. Many systems supply a date service routine which formats the date in a different style. Conversion to the SUPES format should be straightforward.

Most systems provide a time of day service routine which formats the time in the desired style. Some systems also return fractional seconds which can easily be trimmed off. In any case, the format specified by EXTIME must be strictly observed.

EXCPUS is intended to measure performance rather than cost. The quantity returned by EXCPUS should be raw CPU seconds; any weighting for memory use or priority should be removed. I/O time should be included only if it is performed by the CPU.

The hardware ID string for EXPARM should reflect both the manufacturer and model of the processor. For example, 'VAX 8600' rather than just 'VAX' allows the user to make sense of the CPU time returned by EXCPUS.

The software ID string should reflect the release of the operating system in use, such as 'COS 1.11'. It is not a trivial exercise to provide all pertinent information in eight characters for ad hoc systems like CTSS which vary widely between installations. For example, the string 'CFTLIB14' has been used to indicate a variation of the SUPES package for CTSS using CFTLIB and the CFT 1.14 compiler.

On most systems KCSU will give the number of characters per numeric word and KNSU will be unity. For a hypothetical 36-bit processor which allows 8-bit characters to cross word boundaries, KCSU=9 and KNSU=2 would define the storage relationship.

The proper value for IDAU should always be indicated in the reference manual for the compiler where it discusses Unformatted Direct Access files.

The unit/file mode of EXNAME should follow as closely as possible to whatever convention the particular operating system uses for connecting a FORTRAN I/O unit to a file at execution time. This feature should be easy to implement on systems which support preconnection. Support for units 1-99 should be sufficient.

The symbol mode feature of EXNAME should be designed to obtain messages from the system level procedure which activates the program. Eight characters per symbol is a reasonable limit. Support for symbols 0-7 should be adequate.

Support for EXNAME not only requires coding the routine itself, but also designing the system procedure level interface. This interface should always be designed before coding EXNAME. It should fit as cleanly as possible into normal techniques for writing procedures for the system.

The skeleton version of EXUPCS is designed to work on any system which supports lowercase letters. This routine will rarely require any change.

EXREAD must provide a prompt for an interactive device and guarantee that input is echoed. This requires a careful determination of the current execution environment. For example, EXREAD must be able to handle input from a script file as well as from a terminal. Any automatic echo service provided by the operating system should be employed wherever possible, as long as the user supplied prompt appears along with the input data echo.

Most systems provide a FORTRAN callable service routine which returns an address for IXLNUM. In some cases it may be necessary to convert the address to numeric units. For example, addresses on VMS must be multiplied by four to convert from bytes to numeric storage units.

EXMEMY is the most crucial routine in the FORTRAN extension library. It therefore requires a great deal of attention. Care should be taken to ensure that both memory block locations and sizes are measured in numeric storage units. Most systems will round up memory requests to a system defined block boundary; EXMEMY should determine the precise amount of memory allocated. It is generally unnecessary to keep track of memory blocks allocated via EXMEMY; the memory manager can be counted on to perform this task. Release of memory should not be attempted until a great amount of confidence in the implementation of EXMEMY is gained since this affects the cost of memory management, but not performance.

5.3.2 Extension Library Test Program

A short program which exercises all features of the SUPES FORTRAN extension library is available. This program should be considered a starting point for testing a new implementation. Other tests which more extensively exercise complex modules, such as EXMEMY, should be developed as needed.

5.4 INSTALLATION DOCUMENTATION GUIDELINES

A supplement to this document should be written for each operating system on which SUPES is installed. As a minimum, this supplement should include:

1) How to access the SUPES library and link it to an applications program. Individual copies of SUPES should never be propagated as this reduces the quality assurance level of SUPES.

2) How to interface from the operating system to EXNAME for both unit/file mode and symbol mode.

3) How to interface to EXREAD via an interactive device. Information such as how to signal an end of file should be specified.

4) Any known bugs or idiosyncrasies.

The installation supplements for several operating systems are included in Appendix A.

REFERENCES

1. **American National Standard Programming Language FORTRAN**, American National Standards Institute, Inc., ANSI X3.9-1978, New York, 1978.

APPENDIX A

This appendix contains a supplement for each site at which SUPES is currently installed. Changes to the current systems and the addition of new sites will require that this appendix be amended; the information contained here should be considered just a starting point.

All system independent source code for SUPES is stored on the SNLA Central File System under the root directory "/SUPES" in SNLA Standard Text Format. The table below documents the files stored in this directory.

Node	Contents
FRE_FLD.STX	Free field reader source code
MEM_MGR.STX	Memory manager source code
EXT_LIB.STX	Skeleton FORTRAN extension library source code
FRR_TEST.STX	Free field reader test program source code
MEM_TEST.STX	Memory manager test program source code
EXT_TEST.STX	FORTRAN extension library test program source code

These files may be retrieved via the MASS utility and converted to Native Text Format via the NTEXT utility. Sandia personnel may consult the Computer Consulting and Training Division (2614) for details on these utilities.

SITE SUPPLEMENT FOR 1500 VAX CLUSTER (VAX/VMS 4.3)

Linking:

The SUPES package is accessed on the 1500 VAX CLUSTER (SAV01 8600, SAV03 8650, and SAV08 11/785) as an object library located via a system logical name. SUPES routines are linked to an application program as follows:

```
$ LINK your_program,SUPES/LIB,etc.
```

Defining unit/file or symbol/value for EXNAME:

A file name is connected to a unit number via a logical name of the form FORnnn, where "nnn" is a three digit integer indicating the FORTRAN unit number. For example:

```
$ ASSIGN CARDS.INP FOR007
```

causes the following FORTRAN statements to open 'CARDS.INP' on unit 7.

```
      CALL EXNAME( 7, NAME, LN )
      OPEN( 7, FILE=NAME(1:LN) )
```

EXNAME looks for a DCL symbol of the form EXTnn, where "nn" is a two digit integer which defines a symbol number. For example:

```
$ EXT01 = "HELLO"
```

will cause the following call to return NAME='HELLO' and LN=5.

```
      CALL EXNAME( -1, NAME, LN )
```

Interface to EXREAD:

EXREAD will read from SYS$INPUT and automatically echo to SYS$OUTPUT. EXREAD supports all the VMS command line editing features (e.g., CTRL/U, <up-arrow>, etc.). An end-of-file from the terminal keyboard is indicated by CTRL/Z.

Source code:

The source code for the FORTRAN extension library for the VAX/VMS operating system is stored in the SNLA Central File System under node "/SUPES/VMS/EXT_LIB.STX" in SNLA Standard Text format.

Appendix

SITE SUPPLEMENT FOR SNLA CRAY-1/S (COS 1.11)

Linking:

The SUPES package is accessed on the SNLA CRAY-1/S as an object library. The permanent dataset containing SUPES is accessed as follows:

```
ACCESS,DN=SUPES,ID=ACCLIB.
```

SUPES routines are then linked to an application program as follows:

```
LDR,other_options,LIB=SUPES:other_libraries.
```

Defining unit/file or symbol/value for EXNAME:

A file name is connected to a unit number via an alias of the form FTnn, where "nn" is a two digit integer indicating the FORTRAN unit number. For example:

```
ASSIGN,DN=CARDS,A=FT07.
```

causes the following FORTRAN statements to open 'CARDS' on unit 7.

```
  CALL EXNAME( 7, NAME, LN )
  OPEN( 7, FILE=NAME(1:LN) )
```

If no file has been assigned the alias for a particular unit, EXNAME will return a file name of the form TAPEnn, where "nn" is a one (if less than ten) or two digit integer indicating the FORTRAN unit number.

EXNAME looks for a JCL symbol of the form Jn, where "n" is a one digit integer which defines a symbol number. For example:

```
SET(J1='HELLO')
```

will cause the following call to return NAME='HELLO' and LN=5.

```
CALL EXNAME( -1, NAME, LN )
```

Interface to EXREAD:

EXREAD will read from $IN and automatically echo to $OUT. COS at SNLA has no interactive capability.

Known problems:

The CFT 1.11 support routines contain a bug which may cause FREFLD to function improperly. FREFLD was modified for this installation such that application programs which call FREFLD should not notice any problem.

The problem is that the CFT 1.11 support routines do not return an error in the IOSTAT argument for invalid real formats; a zero value and a zero (success) status are returned in such a case. The symptom observed from FREFLD is that KVALUE will indicate that a valid REAL value was specified for a data field which contains an invalid REAL format; the value returned in RVALUE for this field will be set correctly to zero. To work around this problem FREFLD was modified to downgrade KVALUE from one (valid REAL value) to zero (invalid REAL value) under the following conditions:

1) The field does not contain a valid INTEGER value.
2) The REAL value translated for the field is zero.
3) The field does not begin with '0.' nor '.0'.

Source code:

The source code for the FORTRAN extension library for the COS 1.11 operating system is stored in the SNLA Central File System under node "/SUPES/COS/EXT_LIB.STX" in SNLA Standard Text format. The source code for the modified version of FREFLD described above is stored under node "/SUPES/COS/FRE_BUG.STX" in SNLA Standard Text format.

SITE SUPPLEMENT FOR SNLA CRAY X-MP/24 (CTSS/CFTLIB 1.11 or 1.14)

Linking:

The SUPES package is accessed on the SNLA CRAY X-MP/24 as an object library which is stored in a public library file. Two versions of this object library exists: one for the CFT 1.11 compiler, and one for the CFT 1.14 compiler. The CFT 1.11 object library is obtained interactively as follows:

```
lib acclib
ok. x supes11
ok. end
switch supes11 supes
```

Either compiler version can also be obtained within a CCL procedure. For example, the CFT 1.14 object library can be extracted by:

```
lib acclib
-x supes14
-end
switch supes14 supes
```

The SUPES routines are then linked to an application program as follows:

```
ldr other_options,lib=(supes,other_libraries)
```

Note that CFTLIB is a dependent library of SUPES, so there is no need to specify cftlib in the above lib list.

Defining unit/file or symbol/value for EXNAME:

A file name is connected to a unit number via a name of the form tapenn, where "nn" is a one (if less than ten) or two digit integer indicating the

FORTRAN unit number. This name can be replaced via the execution line as shown in the following example:

```
     myprog tape7=cards
```

The above command would cause the following FORTRAN statements within 'myprog' to open 'cards' on unit 7:

```
        CALL EXNAME( 7, NAME, LN )
        OPEN( 7, FILE=NAME(1:LN) )
```

EXNAME looks for a symbol on the execution line of the form extn, where "n" is a one digit integer which defines a symbol number. For example:

```
     myprog ext1=HELLO
```

will cause the following call within 'myprog' to return NAME='HELLO' and LN=5.

```
        CALL EXNAME( -1, NAME, LN )
```

<u>Interface to EXREAD:</u>

EXREAD will read from "input" and automatically echo to "output". By default, EXREAD connects both "input" and "output" to "tty". CTSS defines "tty" as the next higher level controller, which is normally the terminal keyboard / screen for an interactive job, or the JCI / log files for a batch job. An end-of-file from the terminal keyboard is indicated by a null response (just a carriage return).

The default connections for either "input" or "output" can be overridden on the execution line as follows:

```
     myprog input=deck output=list
```

Known problems:

Contrary to the ANSI FORTRAN standard, CTSS does **not** automatically open the standard input and output devices. This causes reading from or writing to UNIT=* to fail unless you add some CTSS-specific code, such as a PROGRAM statement argument list. EXNAME and EXPARM, as well as EXREAD, explicitly open the standard input and output devices according to the rules described above. This is an advantage to the applications programmer since it avoids nonstandard code, but it places the following restrictions on any program which calls EXNAME, EXPARM, or EXREAD under CTSS:

1) Do not use a PROGRAM statement argument list.

2) Do not read from nor write to UNIT=* before a call to either EXNAME, EXPARM, or EXREAD.

Source code:

The source code for the FORTRAN extension library for the CTSS/CFTLIB/SNLA operating system is stored in the SNLA Central File System under nodes "/SUPES/VMS/EXT_111.STX" and "/SUPES/VMS/EXT_114.STX" in SNLA Standard Text format for the CFT 1.11 and 1.14 compilers, respectively.

Distribution:

1265	J. P. Quintenz
1510	J. W. Nunziato
1511	G. G. Weigand (5)
1511	D. K. Gartling (5)
1520	D. J. McCloskey
1521	R. D. Krieg
1521	D. S. Preece
1522	R. C. Reuter, Jr.
1522	C. R. Adams
1522	T. D. Blacker
1523	J. H. Biffle
1523	Z. E. Beisinger
1523	D. P. Flanagan (10)
1523	A. P. Gilkey
1523	J. R. Koteras
1523	L. M. Taylor (10)
1524	A. K. Miller
1524	W. C. Mills-Curran (30)
1530	L. W. Davison
1531	S. L. Thompson
1540	W. C. Luth
1636	W. L. Oberkampf
2614	A. R. Iacoletti
2640	E. J. Theriot
2641	M. R. Scott
2644	R. E. Jones
2645	W. F. Mason
3141-1	S. A. Landenberger (5)
3151	W. L. Garner (3)
3154-1	C. H. Dalin (28--for DOE/OSTI)
8024	P. W. Dean