

SAND90-0247  
Unlimited Release  
Printed May 1990  
(Translation needs work)

Distribution  
Category UC-805

# **SUPES Version 2.1**

## **A Software Utility Package for the Engineering Sciences**

John R. Red-Horse  
Applied Mechanics Division IV  
Sandia National Laboratories  
Albuquerque, New Mexico 87185

William C. Mills-Curran<sup>\*</sup>  
Dennis P. Flanagan<sup>†</sup>

### **Abstract**

The Software Utilities Package for the Engineering Sciences (SUPES) is a collection of subprograms which perform frequently used non-numerical 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.

---

\* Currently employed by Hibbitt, Karlsson & Sorenson, Inc., 100 Medway St., Providence, RI

† Currently employed by Hibbitt, Karlsson & Sorenson, Inc., 100 Medway St., Providence, RI

Intentionally Left Blank

# Table of Contents

<b>1</b>	<b>INTRODUCTION</b>	<b>5</b>
<b>2</b>	<b>INSTALLATION PROCEDURE</b>	<b>7</b>
2.1	VAX/VMS Installation Procedure	7
2.1.1	Building SUPES	7
2.1.2	Building the Test Programs	7
2.1.3	Installing SUPES On Your VMS System	8
2.2	General UNIX Installation Procedure	8
2.2.1	Building SUPES	8
2.2.2	Building the Test Programs	9
2.2.3	Installing SUPES On Your UNIX System	9
<b>3</b>	<b>FREE FIELD INPUT</b>	<b>11</b>
3.1	Keyword/Value Input System	11
3.2	Syntax Rules	12
3.3	Free Field Input Routines	13
3.3.1	External Input Routine (FREFLD)	13
3.3.2	Internal Input Routine (FFISTR)	15
3.3.3	Basic Examples	16
3.4	Utility Routines	17
3.4.1	Get Literal Input Line (GETINP)	17
3.4.2	Strip Leading/Trailing Blanks (STRIPB)	18
3.4.3	Process Quoted String (QUOTED)	19
<b>4</b>	<b>MEMORY MANAGER</b>	<b>21</b>
4.1	Indexing System	21
4.2	Basic Routines	22
4.2.1	Initialize (MDINIT/MCINIT)	22
4.2.2	Define Dynamic Array (MDRSRV/MCRSRV)	23
4.2.3	Delete Dynamic Array (MDDEL/MCDEL)	23
4.2.4	Reserve Memory Block (MDGET/MCGET)	23
4.2.5	Release Unallocated Memory (MDGIVE/MCGIVE)	24
4.2.6	Obtain Statistics (MDSTAT/MCSTAT)	24
4.2.7	Print Error Summary (MDEROR/MCEROR)	24
4.2.8	Enable data initialization (MDFILL/MCFILL)	25
4.2.9	Cancel Data Initialization (MDFOFF/MCFOFF)	26
4.2.10	Basic Example	26
4.3	Advanced Routines	26
4.3.1	Rename Dynamic Array (MDNAME/MCNAME)	26
4.3.2	Adjust Dynamic Array Length (MDLONG/MCLONG)	27
4.3.3	Locate Dynamic Array (MDFIND/MCFIND)	27
4.3.4	Compress Storage (MDCOMP/MCCOMP)	27
4.3.5	Error Flag Query (MDERPT/MCERPT)	28
4.3.6	Modify Error Count (MDEFIX/MCEFIX)	28

4.3.7	Report Last Error (MDLAST/MCLAST) .....	28
4.3.8	Enable Deferred Memory Mode (MDWAIT/MCWAIT) .....	28
4.3.9	Execute Deferred Memory Requests (MDEXEC/MCEXEC) .....	29
4.3.10	Report storage information (MDMEMS/MCMEMS) .....	29
4.4	Development Aids .....	30
4.4.1	List Storage Tables (MDLIST/MCLIST) .....	30
4.4.2	Print Dynamic Array (MDPRNT/MCPRNT) .....	30
4.4.3	Debug Printing (MDDEBG/MCDEBG) .....	30
<b>5</b>	<b>EXTENSION LIBRARY .....</b>	<b>33</b>
5.1	User Interface Routines .....	34
5.1.1	Get Today's Date (EXDATE) .....	34
5.1.2	Get Time of Day (EXTIME) .....	35
5.1.3	Get Accumulated Processor Time (EXCPUS) .....	35
5.1.4	Get Operating Environment Parameters (EXPARM) .....	35
5.1.5	Get Unit File Name or Symbol Value (EXNAME) .....	36
5.2	Utility Support Routines .....	36
5.2.1	Convert String to Uppercase (EXUPCS) .....	36
5.2.2	Prompt/Read/Echo Input Record (EXREAD) .....	36
5.2.3	Evaluate Numeric Storage Location (IXLNUM) .....	37
5.2.4	Evaluate Character Storage Location (IXLCHR) .....	37
5.2.5	Get/Release Memory Block (EXMEMORY) .....	37
5.3	Skeleton Library .....	37
5.3.1	Skeleton Routine Specifications .....	38
<b>6</b>	<b>SUPPORT PROGRAMMER'S GUIDE .....</b>	<b>39</b>
6.1	Free Field Input .....	39
6.1.1	Implementation Notes on FREFLD .....	39
6.1.2	Test Program for FREFLD .....	40
6.2	Memory Manager .....	41
6.2.1	Table Architecture and Maintenance .....	41
6.2.2	Non-ANSI FORTRAN Assumptions .....	42
6.2.3	Standard FORTRAN Implementation .....	42
6.2.4	Test Program .....	42
6.3	Extension Library Implementation .....	43
6.3.1	Implementation Notes for Modules .....	44
6.3.2	Extension Library Test Program .....	46
6.4	Installation Documentation Guidelines .....	46
<b>7</b>	<b>References .....</b>	<b>49</b>
<b>8</b>	<b>SITE SUPPLEMENTS .....</b>	<b>51</b>

# 1 INTRODUCTION

The Software Utilities Package for the Engineering Sciences (SUPES) is a collection of subprograms which perform frequently used non-numerical 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<sup>1</sup>, while the subprograms in category three are written in the C programming language. Thus providing a standardized FORTRAN interface to several system dependent features across a variety of hardware configurations while using a single set of source files. This feature can be viewed as a maintenance aid from several perspectives. Among these are: there is only one set of source files to account for, it allows one to standardize the build procedure, and it provides a clearer starting point for any future ports. In fact, a build procedure is now part of the standard SUPES distribution and is documented in Chapter 2 . Further, the system dependent modules set an appropriate template for the porting of SUPES to other hardware and/or software configurations.

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:

- Providing a library of useful subprograms.
- Defining a standard interface format for common utilities.
- 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 5 ), 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.

It is the intention of the authors to maintain SUPES on all scientific computer systems commonly used by Engineering Sciences Directorate (1500) staff. Currently these systems include:

1. Sun 3 and Sun 4 running SunOS operating system version 4.0.3 and later,
2. VAXen running VMS version 4.5 and later,
3. Cray X/MP and Y/MP running UNICOS version 5.0 and later, and
4. Alliant F/X 8 running Concentrix 5.0.0.

A notable omission to the above list is the Cray running either CTSS or the COS operating systems. These configurations still require the FORTRAN source code for the extension library that was provided in previous implementations of SUPES [SUPES]. This code continues to be included in the current standard SUPES distribution, though a build procedure designed for these systems is not. Specific ports of the SUPES utilities to new machines and/or operating systems will be added to the original source files as the need arises. Other Sandia personnel may obtain copies of SUPES from the authors. SUPES will also be available to non-Sandia personnel through the National Energy Software Center.

## 2 INSTALLATION PROCEDURE

SUPES now contains a codified procedure for installing it as a part of the standard distribution.

### 2.1 VAX/VMS Installation Procedure

#### 2.1.1 Building SUPES

Under normal conditions, the VMS version of the SUPES distribution will come in the form of a BACKUP saveset, SUPES2\_1.BCK. The installer should set the default directory to a suitable place and unbundle the saveset as follows:

```
$ BACKUP SUPES2_1.BCK/SAVESET [ ]
```

Then, set default to [ .SUPES2\_1.BUILD ], and execute the build procedure by entering the command:

```
$ @BUILD_VMS
```

and wait for the build to be performed. You will be prompted for a message to include in an update file (UPDATE.QA). Do this by entering a message between a pair of double quotes (") followed by a carriage return, <CR>. The library will be built as SUPES2\_1.OLB in the [ .BUILD ] directory. IMPORTANT!!! Version 2.3 of VAX C running v4.5 of VMS exhibits a strange bug: when compiling the module [ .EXT\_LIB.PORTABLE ] EXDATE.C in the command procedure BUILD\_VMS.COM it doesn't find one of the "header" modules and bombs with an error. The net result is that you will have to copy this one to the build directory yourself, compile it with CC, and add it to the library. Here are the commands to do that:

```
$ COPY [-.EXT_LIB.PORTABLE]EXDATE.C [ ] ! from the build subdirectory
$ CC EXDATE
$ LIBRARY/REPLACE/LOG SUPES2_1 EXDATE
```

#### 2.1.2 Building the Test Programs

Once you have done the installation, there is a set of test procedures that exercise each of the SUPES capabilities separately. They are located in the top-level directory and are named: EXTTEST.F, MEMTEST.F, and FFRTEST.F. To build these, use the command procedure, BUILD\_TESTS.COM, which is invoked with:

```
$ @BUILD_TESTS
```

Once you have done this step, each of the test procedures will be available for use in the [ .BUILD ] subdirectory. To use any one of them refer to the proper section in Chapter [Ref: sec:support] as well as to a file titled OUTPUT.LIS located in the individual subdirectories, [ .EXT\_LIB ], [ .FRE\_FLD ], or [ .MEM\_MGR ]. Finally, you can refer to the source files for the test procedures themselves.

### 2.1.3 Installing SUPES On Your VMS System

As a last step, install the SUPES library on your VMS system. This is done by running the command procedure, `VMSINSTALL.COM`. It should copy the the library to the directory of your choice, and set up the required logicals. Some things to note:

1. To perform the operations in the `VMSINSTALL.COM` command procedure, you will be required to have `SYSTEM` privileges.
2. You may want to have your system manager look at this file and insert some sections of it in a system startup command procedure. Otherwise, the appropriate definitions will be lost when the system is rebooted.
3. If you don't have the required privileges, you should edit `VMSINSTALL.COM` to remove any qualifier that requires them and invoke this newly created version in your `LOGIN.COM`. This will set up the logical names in your process name table and allow you to use SUPES as described in this manual.

## 2.2 General UNIX Installation Procedure

### 2.2.1 Building SUPES

The general build scheme for all of the UNIX derived operating systems will be done through the `make` utility. This procedure should help the maintainer deal with any upgrades, bug fixes, etc. The distribution itself will generally be distributed as UNIX `tar` file named `supes2_1.tar`. To install SUPES, go to the directory that you want to have as a parent of the SUPES tree and unbundle the distribution. In the example below, the directory `/usr/local` has been arbitrarily chosen as this parent—individual sites have the option of choosing a different location, depending on their conventions. An example of the required command sequence follows [Footnote: Here, and throughout the remainder of this manual, the UN\*X interaction will be documented as follows: the user prompt will be `\%`, comments will be offset by `<--`, and the text in between will denote the user supplied commands.] :

```
\% cd /usr/local      <-- ``/usr/local`` will contain the distribution.
\% tar xf supes2_1.tar <-- If you get your distribution via tape, man tar.
\% cd supes2_1
```

You will now be in the top-level directory of the distribution; each directory reference from this point onward will be made relative to this directory. If a makefile exists for your system named `makefile.$(ARCH)` in any of the source directories—`./ext_lib/portable`, `./fre fld`, or `./mem_mgr` then a machine specific makefile has been written. For example, under UNICOS, the file `makefile.unico` exists in `./mem_mgr`, so in order to do the build for that system one would need to perform the following command:

```
\% make ARCH=.unico
```

from the `supes2_1` directory and the build will proceed. At this point, you will be prompted for a message to add to the update file (`update.qa` in the `./build` subdirectory). Conclude this message with a `^D` (i.e., input a “D” while holding down the

Control key simultaneously) at the beginning of a line. The sequence will look something like this:

```
% Enter Message for Update File (./build/update.qa).
% End with a CNTL-D On A New Line.
% Initial UNICOS build.          <---/ Lines input by user
% ^D                             <---/
```

The archived library, titled `libsupes.a`, will be built in the “./build” subdirectory.

There are a couple of things to note: the “.” in the above make statement *IS* significant!!!! Further, the file name has a suffix “unico” due to the fact that Cray UNICOS restricts file names to be fewer than fifteen characters.

In the event that such a makefile does *NOT* exist then one of two things is true. Either typing the simple command:

```
% make
```

from the `supes2_1` directory will suffice, or an appropriate makefile does not exist. In the former case you are done, while in the latter, the consequences are much greater. More to the point, it probably means that the code will not run on your machine without modification. If this is the case, you will need to port the C source files in the directory `./ext_lib/portable`. Use the existing source as a guide and reference the “README” file in this directory.

### 2.2.2 Building the Test Programs

In most instances, once you have done the installation, you have also built the set of test procedures that exercise each of the SUPES capabilities separately (cf. Chapter [Ref: sec:support] ). They are located in the top-level `supes2_1` directory and are named: `extttest`, `memttest`, and `ffrttest`. Look for them in the current directory. If they’re not there, then something has happened to prevent the test procedures from being built after the actual build of the SUPES library and you will be required to build them manually. Doing this is a system dependent problem; here’s how you would go about building `extttest` on the Alliant:

```
% fortran -o extttest extttest.f build/libsupes.a
```

Or, on the Cray under UNICOS with the `cf77` compiler it’s:

```
% cf77 -o extttest extttest.f build/libsupes.a
```

To use each of the programs, refer to the proper section in Chapter [Ref: sec:support] as well as to the individual subdirectories for a file titled, `output.lis` and finally, refer to the source files for the test procedures themselves.

### 2.2.3 Installing SUPES On Your UNIX System

As a last step, install the SUPES library in a suitable place on your UNIX system. To do this, just enter the command

```
% make install
```

from the `supes2_1` directory. You should note that the proper permission will be required to place the library in its final resting place (the default is `/usr/local/lib`).

## 3 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 [ansi].
4. The interface to the applications program must be clear and flexible.

Version 2 of the SUPES free field reader differs from version 1 in the following areas:

1. An interface has been provided to allow character strings to be input to the free field reader in addition to reading from files. This allows the applications programmer to perform more sophisticated string parsing than would be possible when reading only from a file.
2. Whole, real numbers (e.g., 12.3E3) will translate to both INTEGER and REAL values if the absolute value of the number is not greater than 1.0E9.
3. Quoted strings are allowed. This makes the free field reader more compatible with the standard FORTRAN free field input. No interpretation of characters (except for internal quotes) is performed within a quoted string.

### 3.1 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, “5.2345E3” 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 four syntax markers in SUPES: field separators which delimit data fields, a quote character which encloses literal strings, 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 use 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.

### **3.2 Syntax Rules**

The syntax rules for the SUPES free field input structure are listed below. This syntax describes how input records/strings 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 non-data character or the end of the input line. A non-data 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/string will be treated as a continuation of the current line. All characters after and including the continuation indicator on the current line are ignored. Multiple records/strings that are “joined” by continuation indicators are treated as a single logical record.
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 not contained in a quoted string are converted to uppercase. All other non-printable ASCII 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. A quoted string is a data field in which the quote (') character is the first nonblank character. An internal quote is indicated with 2 consecutive quote characters. If an end quote character is not included, then the remainder of the record (excluding any trailing blanks) is treated as part of the quoted string. Within a quoted string, no character conversion to uppercase is performed. Delimiters (other than quotes) are treated as part of the string. Interpretation of data to numeric data will be performed, if possible.
9. A data field which does not begin with the quote character, but has a quote internal to the field (e.g., MOM'S) is not considered a quoted string. In this case, the internal quote is not a special character.
10. The maximum length of an input record (FREFLD only) is 132 characters. Input strings to FFISTR may be any length.

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.

### 3.3 Free Field Input Routines

The user interface to the SUPES free field input system consists of two subroutines: FREFLD and FFISTR. Both routines perform parsing functions of strings. The main difference is that FFREFLD gets its input from a FORTRAN I/O unit while FFISTR gets its input from a character string. In fact, FREFLD uses FFISTR to perform parsing functions once FREFLD has read a record.

#### 3.3.1 External Input Routine (FREFLD)

Input is prompted for, read, and echoed via FREFLD 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 [Ref: sec:getinp] , while the parsing is performed by FFISTR.

The arguments to FREFLD are prescribed below.

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

**KININTEGER** Read Only 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 non-zero, the caller is responsible for opening/closing this unit.

**KOUTINTEGER** Read Only Unit to which to echo input. If zero, do not echo other than to the standard output device as described above. If non-zero, the caller is responsible for opening/closing this unit.

**PROMPTCHARACTER** \last (\last ) Read Only 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.

**MFIELDINTEGER** Read Only Maximum number of data fields to be returned. The dimensions of each of the output arrays described below must be greater than or equal to this number.

**IOSTATINTEGER** Write Only ANSI FORTRAN I/O status:

IOSTAT	<	0	End of File
IOSTAT	=	0	Normal
IOSTAT	>	0	Error

**NFIELDINTEGER** Write Only Number of data fields found on this logical record. 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.

**KVALUEINTEGER** Array Write Only Translation states of the data fields. The value of each element of this array is interpreted as follows:

KVALUE	Meaning
-1	This is a null field.
0	This is a non-numeric 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.

**CVALUECHARACTER** \last (\last ) Array Write Only 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.

IVALUEINTEGER ArrayWrite Only 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.

RVALUEREAL ArrayWrite Only 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.

### 3.3.2 Internal Input Routine (FFISTR)

Internal input (i.e., a character string) is parsed via FFISTR using character strings supplied through FFISTR's argument list. FFISTR returns the parsed data field values defined in the input string. If a string contains a continuation character, a flag is returned to the user indicating that another string should be supplied to complete the logical record. The arguments to FFISTR are prescribed below.

```
CALL FFISTR( LINE, MFIELD, IDCONT, NFIELD, KVALUE,  
*           CVALUE, IVALUE, RVALUE )
```

LINECHARACTER\last (\last )Read Only Input string. This argument contains the data to be parsed.

MFIELDINTEGERRead Only Maximum number of data fields to be returned. The dimensions of each of the output arrays described below must be greater than or equal to this number.

IDCONTINTEGERRead and Write Continuation flag. 0 means no continuation. On input, this flag indicates if the previous string contained a continuation indicator. In this case, the current string will be treated as part of the same logical record as the previous string.

NFIELDINTEGERWrite Only Number of data fields found on this logical record. 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.

KVALUEINTEGER ArrayWrite Only Translation states of the data fields. The value of each element of this array is interpreted as follows:

KVALUE	Meaning
-1	This is a null field.
0	This is a non-numeric 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.

CVALUECHARACTER\last (\last ) ArrayWrite Only 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.

IVALUEINTEGER ArrayWrite Only 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.

RVALUEREAL ArrayWrite Only 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.

### 3.3.3 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	2	2._____	2.00	2
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

INPUT RECORD:

'Quoted strings', '5 ', '\$\*,=''' \$ rest is comment

Results returned from FREFLD: NFIELD = 3

I	KVALUE(I)	CVALUE(I)	RVALUE(I)	IVALUE(I)
1	0	Quoted_strings__	0.000E+00	0
2	2	5_____	50.0	50
3	0	\$*,='''_____	0.000E+00	0
4	-1	_____	0.000E+00	0
5	-1	_____	0.000E+00	0

INPUT RECORD:

quotes's

Results returned from FREFLD: NFIELD = 1

I	KVALUE(I)	CVALUE(I)	RVALUE(I)	IVALUE(I)
1	0	QUOTES'S_____	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

### 3.4 Utility Routines

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

#### 3.4.1 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 )
```

**KININTEGER** Read Only 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). non-zero, the caller is responsible for opening/closing this unit.

**KOUTINTEGER** Read Only Unit to which to echo input. If zero, do not echo other than to the standard output device as described above. If non-zero, the caller is responsible for opening/closing this unit.

**PROMPTCHARACTER**\last (\last ) Read Only 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.

**LINECHARACTER**\last (\last ) Write Only Line of input. This string will be blank-filled or truncated, if necessary. The length of the string is set by the caller, but should not exceed 132.

**IOSTATINTEGER** Write Only ANSI FORTRAN I/O status:

IOSTAT	<	0	End of File
IOSTAT	=	0	Normal
IOSTAT	>	0	Error

### 3.4.2 Strip Leading/Trailing Blanks (STRIPB)

This routine is called by FREFLD and FFISTR 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 non- blank characters. If a substring is passed, these locations are relative to 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 )
```

STRINGCHARACTER\last (\last )Read Only Any character string.

ILEFTINTEGERWrite Only Relative index of the first non-blank character in STRING. ILEFT = LEN(STRING) + 1 if STRING = ' '.

IRIGHTINTEGERWrite Only Relative index of the last non-blank character in STRING. IRIGHT = 0 if STRING = ' '.

### 3.4.3 Process Quoted String (QUOTED)

This routine is called by FFISTR to remove the delimiting quotes from a quoted string. It also converts any repeated quotes into single quotes. (This is a common method for indicating internal quotes.)

The arguments to QUOTED are prescribed below.

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

STRINGCHARACTER\last (\last )Read and Write Any character string. On output, the first and last quotes are removed, and internal (repeated) quotes are converted to single quotes. If the trailing quote is omitted, then the remainder of the input record (excluding trailing blanks) is considered part of the quoted string.

ILEFTINTEGERWrite Only Relative index of the first character in the string. This is always the location of the first character inside the leading quote.

IRIGHTINTEGERWrite Only Relative index of the last character in STRING. IRIGHT = 0 if the quoted string is null.

Intentionally Left Blank

## 4 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, LOGICAL and CHARACTER type arrays.

Because the array sizes in most programs are problem dependent, a 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.

In SUPES, 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 extension library (Chapter [Ref: sec:extlib] ). All memory requests are in terms of *numeric storage units* for numeric data (integer, real, or logical) and *character storage units* for character data [ansi].

An important design feature of the memory manager is that the memory manager can be supported even when the system-dependent dynamic memory request routines are not implemented on a system. In this case, the memory manager will operate, allocating space from a user-supplied work array. This mode is described as dynamic allocation of static memory. Thus, modification of a user's application program is minimal on systems where dynamic memory is not implemented.

All user entry points to memory manager routines begin with either "MD" or "MC." In most cases, the "MD" routines are used for numeric data, while the "MC" routines are for character data. In some cases, however, the routines are interchangeable. These routines are documented as synonyms.

In this document, the term "Mx" is used to refer simultaneously to both "MD" and "MC" routines. Thus, MxRSRV is a reference to both MDRSRV and MCRSRV subroutines.

The memory manager utility is divided into three categories; basic routines, advanced routines, and development aids. These categories will be discussed in sections [Ref: sec:mbas] through [Ref: sec:mdev] .

### 4.1 Indexing System

In order to use the memory manager properly, the user must first understand the concept of using a base array with indices for accessing memory address locations. At the core of this concept is FORTRAN's convention of passing SUBROUTINE array references by address. The memory manager references all memory addresses relative to the addresses of user-supplied base arrays—one each for numeric and character data. A reference to memory is made in terms of a pointer to these base arrays. Specifically, the memory man-

ager determines an indexing parameter by first determining the offset of the appropriate memory location relative to the address of the correct base array. The index is then computed in terms of the proper storage units (either character or numeric). Note that the resulting indexes may take on a wide range of values, including negative numbers.

The base arrays must comply with the following rules:

1. Numeric base arrays *must* be of type INTEGER, REAL, or LOGICAL. Modified word length storage arrays such as INTEGER\last 2 or REAL\last 8 will result in invalid indexes with no error message.
2. Character base arrays *must* be declared CHARACTER\last 1.

The following FORTRAN statements define valid base arrays:

```
DIMENSION NUMBAS(1)
CHARACTER*1 CHRBAS(1)
```

Only one base array from each category (numeric and character) 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 offsetting index to a subprogram. For example, for a base arrays NUMBAS and CHRBAS and indexes IP1 and IP2, a subroutine call would be:

```
CALL SUBBIE ( NUMBAS(IP1), CHRBAS(IP2) )
```

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.

## 4.2 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.

### 4.2.1 Initialize (MDINIT/MCINIT)

The memory manager *must* be initialized with a calls to MDINIT and MCINIT before any memory can be allocated. The main purpose of the initialization is to determine the location of the numeric and character base arrays in memory. MDINIT must be called first, and MCINIT second. In the case where character dynamic memory is not used, MCINIT need not be called. When calling MxINIT, the user must pass (explicitly or implicitly) subscript 1 of the base array.

```
CALL MDINIT (NUMBAS(1))
CALL MCINIT (CHRBAS(1))
```

NUMBASINTEGER, LOGICAL or REAL Array or Array ElementRead Only This array is used as a base reference to all dynamically allocated numeric memory.

CHRBASCHARACTER\last 1 Array or Array ElementRead Only This array is used as a base reference to all dynamically allocated character memory.

#### 4.2.2 Define Dynamic Array (MDRSRV/MCRSRV)

MxRSRV declares new dynamic arrays. The user specifies the space required, and an index to the new space is returned. Note that, by default, the contents of the new storage are not initialized to any specific value. MxFILL may be used for data initialization.

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

```
CALL MCRSRV (NAME, NEWIDX, NEWLEN)
```

NAMECHARACTER\last (\last )Read Only 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 beginning with a nonblank are used for comparison. This comparison is case-insensitive and embedded blanks are significant.

NEWIDXINTEGERWrite Only This is the index to storage allocated to this dynamic array relative to the base array. The index for numeric data is to be used with the numeric array supplied to MDINIT, and character data is to be used with the character array supplied to MCINIT.

NEWLENINTEGERRead Only 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 an index equal to one. The value of NEWLEN is in terms of numeric storage units for numeric data and character storage units for character data.

#### 4.2.3 Delete Dynamic Array (MDDEL/MCDEL)

MDDEL and MCDEL release the memory that is allocated to a dynamic array for numeric and character storage, respectively.

```
CALL MDDEL (NAME)
```

```
CALL MCDEL (NAME)
```

NAMECHARACTER\last (\last )Read Only This is the name of the dynamic array which is to be deleted. The array name must match an existing name in the dictionary and be of the correct type (numeric or character) for the operation. The first eight characters beginning with a nonblank are used for comparison. This comparison is case-insensitive and embedded blanks are significant.

#### 4.2.4 Reserve Memory Block (MDGET/MCGET)

MDGET and MCGET reserve a contiguous block of memory without associating the block of memory with an array. MxGET should be called prior to a series of calls to MxRSRV to improve efficiency and to reduce memory fragmentation. Further discussion of the operation of MxGET is found in section [Ref: sec:table] .

```
CALL MDGET (MNGET)
```

```
CALL MCGET (MNGET)
```

MNGETINTEGERRead only This specifies the desired contiguous block size in numeric storage units for MDGET or character storage units for MCGET.

#### 4.2.5 Release Unallocated Memory (MDGIVE/MCGIVE)

MxGIVE causes the memory manager to return unused storage to the operating system, if possible. MDGIVE and MCGIVE are synonyms.

```
CALL MDGIVE ( )  
CALL MCGIVE ( )
```

#### 4.2.6 Obtain Statistics (MDSTAT/MCSTAT)

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

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

**MNERRS**INTEGERWrite Only This is the total number of errors detected by the memory manager during the current execution.

**MNUSED**INTEGERWrite Only This is the total number of storage units that are currently allocated to dynamic arrays. MDSTAT returns the numeric storage in numeric storage units, and MCSTAT returns the character storage in character storage units. If any storage has been requested in the deferred mode and not yet allocated by the memory manager (Section [Ref: sec:wait] ), this storage is counted as though it were actually allocated.

#### 4.2.7 Print Error Summary (MDEROR/MCEROR)

MxEROR prints a summary of all errors detected by the memory manager. The return status of the last memory manager routine called is also printed. MxEROR should be called any time an error is detected by a call to MxSTAT. Table 1 lists the error codes. MDEROR and MCEROR are synonyms.

Several of the error codes listed in Table 1 are not a result of a user error, but are used to signal an internal error, or that an internal array is full. For example, the table which records the names of the arrays allocated with MxRSRV may not be large enough for the application. In this case, the memory manager subroutines must be modified to accommodate the user. A local support person should perform this task.

```
CALL MDEROR (IUNIT)  
CALL MCEROR (IUNIT)
```

**IUNIT**INTEGERRead Only This is the FORTRAN unit number of the output device.

**Table 1. Memory Manager Error Codes**

<i>ERROR CODES</i>	
1	SUCCESSFUL COMPLETION
2	UNABLE TO GET REQUESTED SPACE FROM SYSTEM
3	DATA MANAGER NOT INITIALIZED

**Table 1. Memory Manager Error Codes**

<i>ERROR CODES</i>		
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
15		INVALID ERROR CODE
16		INVALID INPUT NAME
17		ILLEGAL CALL WHILE IN DEFER MODE
18		NAME IS OF WRONG TYPE FOR OPERATION
<b>*These are not user errors</b>		

**4.2.8 Enable data initialization (MDFILL/MCFILL)**

MxFILL defines a fill/initialization pattern that is to be used for newly allocated storage. MDFILL and MCFILL are in effect until canceled by MDFOFF and MCFOFF, respectively. MDFILL and MCFILL operate independently.

CALL MDFILL (NUMDAT)

CALL MCFILL (CHRDAT)

NUMDATINTEGER, REAL or LOGICALRead Only This is the initialization datum for new storage allocated with MDRSRV or extended with MDLONG. The memory manager makes no attempt to identify the type (INTEGER, REAL, or LOGICAL) of either the initialization datum or of a newly allocated array. Instead, the bit of the initialization datum is stored without interpretation. This pattern is then used to initialize new storage. Since the internal machine representation of REAL data is different than INTEGER data (or LOGICAL data), the user may experience unexpected results when dynamic memory is used as a numeric type which is different from the type of the initialization datum.

CHRDATCHARACTER\last (\last )Read Only This is the initialization data for new storage allocated with MCRSRV or extended with MCLONG. Only the first character of CHRDAT is used.

### 4.2.9 Cancel Data Initialization (MDFOFF/MCFOFF)

MDFOFF and MCFOFF cancel the data initialization for numeric and character data, respectively. MDFOFF and MCFOFF operate independently.

```
CALL MDFOFF ( )  
CALL MCFOFF ( )
```

### 4.2.10 Basic Example

```
DIMENSION BASE(1)  
CHARACTER*1 CBASE(1)  
CALL MDINIT (BASE(1))  
CALL MCINIT (CBASE(1))  
CALL MDGET (20)  
CALL MDFILL (0.)  
CALL MCFILL ('Z')  
CALL MDRSRV ('FIRST', I1, 10)  
CALL MDRSRV ('SECOND', I2, 10)  
CALL MCRSRV ('THIRD', I3, 10)  
CALL MDDEL ('SECOND')  
CALL MDGIVE ( )  
CALL MDSTAT (MNERRS, MNUSED)  
IF (MNERRS .NE. 0) THEN  
    CALL MDEROR (6)  
    STOP  
END IF  
CALL SUBPRG (BASE(I1), CBASE(I3))
```

## 4.3 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.

### 4.3.1 Rename Dynamic Array (MDNAME/MCNAME)

MxNAME renames a dynamic array from NAME1 to NAME2. The location of the array is not changed, nor is its length. MDNAME is used for numeric arrays and MCNAME is used for character arrays.

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

NAME1 CHARACTER\last (\last )Read Only This is the old name of the array. The first eight characters after the first nonblank are used for comparison. This comparison is case-insensitive and embedded blanks are significant.

NAME2 CHARACTER\last (\last )Read Only This is the new name of the array. The first eight characters starting from a nonblank are used for the new name. This comparison is case-insensitive and embedded blanks are significant.

### 4.3.2 Adjust Dynamic Array Length (MDLONG/MCLONG)

MxLONG 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 MxLONG invalidates the previous index to this array if the array is extended. MDLONG is used for numeric arrays and MCLONG is used for character arrays.

```
CALL MDLONG (NAME, NEWIDX, NEWLEN)
CALL MCLONG (NAME, NEWIDX, NEWLEN)
```

NAMECHARACTER\last (\last )Read Only This is the name of the dynamic array which the user wishes to extend or shorten.

NEWIDXINTEGERWrite Only This is the new index to the dynamic array.

NEWLENINTEGERRead Only This is the new length for the dynamic array in numeric storage units for MDLONG and in character storage units for MCLONG.

### 4.3.3 Locate Dynamic Array (MDFIND/MCFIND)

MxFIND returns the index and length of storage allocated to a dynamic array. This routine would be used if the index from an earlier call to MxRSRV was not available in a particular program segment. MDFIND is used for numeric arrays and MCFIND is used for character arrays.

```
CALL MDFIND (NAME, NEWIDX, NEWLEN)
CALL MCFIND (NAME, NEWIDX, NEWLEN)
```

NAMECHARACTER\last (\last )Read Only This is the name of the dynamic array to be located.

NEWIDXINTEGERWrite Only This is the index to the dynamic array relative to the user's reference array. Because an index can take any value, the returned value cannot be used as an indication of success or failure of MxFIND. MxSTAT should always be used for error checking.

NEWLENINTEGERWrite Only This is the length of the dynamic array in numeric or character storage units for MDFIND and MCFIND, respectively.

### 4.3.4 Compress Storage (MDCOMP/MCCOMP)

MxCOMP causes fragmented memory to be consolidated. Note that this may cause array storage locations to change. It is important to realize that all indexes must be recalculated by calling MxFIND after a compress operation. A call to MxCOMP prior to MxGIVE will result in the return of the maximum memory to the system. MDCOMP and MCCOMP are synonyms.

```
CALL MDCOMP ( )
CALL MCCOMP ( )
```

### 4.3.5 Error Flag Query (MDERPT/MCERPT)

MxERPT requests the memory manager to report the number of errors accumulated for a particular error flag. A programmer may use this to determine more detailed information than what is available from MxSTAT. MDERPT and MCERPT are synonyms.

```
CALL MDERPT (IFLAG, NERRS)
CALL MCERPT (IFLAG, NERRS)
```

**IFLAGINTEGER**Read Only IFLAG specifies the flag number for which the user wishes an error count. A list of the error flags can be printed by calling MxEROR.

**NERRSINTEGER**Write Only NERRS will contain the error count.

### 4.3.6 Modify Error Count (MDEFIX/MCEFIX)

MxEFIX allows the error count for a particular error flag to be set to a specified value. MDEFIX and MCEFIX are synonyms.

```
CALL MDEFIX (IFLAG, NERRS)
CALL MCEFIX (IFLAG, NERRS)
```

**IFLAGINTEGER**Read Only IFLAG specifies the number of the error flag which will be set. See Table [Ref: tab:ecode] for a list and description of these flags.

**NERRSINTEGER**Read Only NERRS is the new value for the error count.

### 4.3.7 Report Last Error (MDLAST/MCLAST)

MxLAST requests the flag number of the last error. MDLAST and MCLAST are synonyms.

```
CALL MDLAST (IFLAG)
CALL MCLAST (IFLAG)
```

**IFLAGINTEGER**Write Only IFLAG is the number of the last error caused by a previous call to the memory manager.

### 4.3.8 Enable Deferred Memory Mode (MDWAIT/MCWAIT)

MxWAIT enables the deferred memory mode of the memory manager. In this mode, any requests for additional memory with MxRSRV are satisfied only if a system call is not required. If a system call is required, the request for memory is deferred and will be satisfied when the deferred mode is canceled by calling MxEXEC or a call to MxLONG requires a system call for memory for an existing array. MDWAIT and MCWAIT are synonyms.

Because the deferred mode may not actually provide array space at the time a call to MxRSRV is made, the base array pointer returned by MxRSRV may not be valid. In fact, for a deferred request, an invalid index is intentionally returned so that the requested array space cannot be erroneously used. When the deferred memory requests are eventually satisfied (by calling MxEXEC), the indexes are automatically, asynchronously updated by

the memory manager. Thus, upon return from MxEXEC the indexes used in MxRSRV will have a valid value.

The deferred mode is provided as a means of reducing the sometimes time-consuming calls to the operating system for new memory. A similar efficiency could be realized by judicious use of MxGET, but the deferred mode relieves the user of the burden of adding all memory requests before calling MxRSRV. The deferred mode is a sophisticated capability and should not be enabled if the user does not understand its implications.

The deferred mode must be used as follows:

1. The deferred mode begins with a call to MxWAIT.
2. Except for MxPRNT, all memory manager calls are permissible in the deferred mode.
3. Indexes returned by MxRSRV, MxFIND, and MxLONG may not be assigned to other variables while the deferred mode is in effect.
4. Dynamic memory may not be accessed while the deferred mode is in effect.
5. The deferred mode is canceled by calling MxEXEC.

```
CALL MDWAIT ( )
```

```
CALL MCWAIT ( )
```

#### **4.3.9 Execute Deferred Memory Requests (MDEXEC/MCEXEC)**

MxEXEC causes all deferred mode memory requests to be satisfied with a single call to the operating system for the required memory. The deferred mode is also canceled. MDEXEC and MCEXEC are synonyms.

After returning from MxEXEC, all indexes to array space which was deferred are updated.

```
CALL MDEXEC ( )
```

```
CALL MCEXEC ( )
```

#### **4.3.10 Report storage information (MDMEMS/MCMEMS)**

MxMEMS reports numeric or character storage information. This information may be useful for planning storage strategies during code development and for flow control during code execution.

```
CALL MDMEMS (NSUA, NSUD, NSUV, NSULV)
```

```
CALL MCMEMS (NSUA, NSUD, NSUV, NSULV)
```

**NSUAINTEGER**Write Only NSUA is the number of numeric/character storage units currently allocated and not deferred.

**NSUDINTEGER**Write Only NSUD is the number of numeric/character storage units currently deferred.

**NSUVINTEGER**Write Only NSUV is the amount of void space in numeric or character storage units. Note that MDMEMS and MCMEMS may be reporting the same space for NSUV, but in different units.

NSULVINTEGERWrite Only NSULV is the size of the largest void space in numeric or character storage units. Note that MDMEMS and MCMEMS may be reporting the same space for NSULV, but in different units.

## 4.4 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, except as a response to a memory manager error.

### 4.4.1 List Storage Tables (MDLIST/MCLIST)

MxLIST prints the contents of the memory manager's internal tables. Section [Ref: sec:table] describes these tables. MDLIST and MCLIST are synonyms.

```
CALL MDLIST (IUNIT)
CALL MCLIST (IUNIT)
```

IUNITINTEGERRead Only This is the FORTRAN unit number of the output device.

### 4.4.2 Print Dynamic Array (MDPRNT/MCPRNT)

MxPRNT prints the contents of an individual numeric and character array, respectively.

```
CALL MDPRNT (NAME, IUNIT, TYPE)
CALL MCPRNT (NAME, IUNIT, NGRUP)
```

NAMECHARACTER\last (\last )Read Only This is the name of the array to be printed.

IUNITINTEGERRead Only This is the FORTRAN unit number of the output device.

TYPECHARACTER\last (\last )Read Only TYPE 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.

NGRUPINTEGERRead Only NGRUP controls the number of characters that are grouped together without intervening spaces. Since the storage is declared as a CHARACTER\last 1 array, NGRUP allows the user to format the output consistent with longer character strings.

### 4.4.3 Debug Printing (MDDEBG/MCDEBG)

Debug printing causes error messages to be printed by the memory manager at the time an error is detected. The default is no debug printing — errors are detected, but are only reported when requested by MxSTAT and MxERPT. MDDEBG and MCDEBG are synonyms.

```
CALL MDDEBG (IUNIT)
CALL MCDEBG (IUNIT)
```

**IUNITINTEGER** Read Only IUNIT controls debug printing. A negative or zero value turns debug printing off. A positive value of IUNIT will cause any error messages to be printed to FORTRAN unit number IUNIT.

Intentionally Left Blank

## 5 EXTENSION LIBRARY

The SUPES Extension Library provides a uniform interface to necessary operating system functions which are not included in the ANSI FORTRAN-77 standard. While the Extension Library itself is implemented in the C programming language, the interface from a FORTRAN calling program is implemented in the same manner as in previous versions of SUPES [SUPES]. Thus, in the sections below which describe the calling sequence, the calls are defined accordingly. This package makes it possible to maintain many codes on different operating systems with a single point of support for system dependencies. Moreover, this maintenance is done via a single set of source files which should not only reduce the time involved in bookkeeping, but allow for the procedures for building a SUPES library to be codified as well. 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, or to improve execution efficiency.

Each module included in the SUPES 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.

The SUPES Extension Library routines are designed to minimize the effort required to implement this software on a new operating system. This is especially true given that the current single set of source files handle a variety of system architectures and software configurations, making those files useful as starting points for a new port. Operating system dependencies have been isolated at the lowest possible level with the major difficulty of a specific port being that of supplying the proper FORTRAN interface with each C subprogram module. To make the above comments more concrete, consider the following section of code excerpted from the source file `exdate.c`:

```
#include <errno.h>

#if defined (unix)
# if defined (alliant)

#   include <sys/types.h>
#   include <sys/time.h>
exdate_( string )           /* Sadly, on the Alliant, */
                             /* strings are not passed */
                             /* similar to the SUN. */

char *string;

# endif                       /* Alliant */
# if defined (sun)

#   include <sys/time.h>
exdate_( string, len )
char *string;
```

```

        long len;

# else                                /* Not Sun */
#   if defined (CRAY)

#     include <sys/types.h>
#     include <time.h>
#     include <fortran.h>
        EXDATE( string )
        _fcd string;

#   endif                              /* Unicos */
# endif                                /* Sun */
#else                                  /* Not UNIX */
#   if defined (VMS)

#include time
#include descrip
        exdate( string )
        struct dsc$descriptor_s *string; /* We know that the VAX
saves */

                                                /* FORTRAN char arrays */
                                                /* as descriptors. */

#   else                                /* not VMS */
#   endif                              /* VMS */
#endif                                /* UNIX */

```

The passages beginning with `#if defined` query the system at compile time through the use of a pre-processor to determine the hardware/software configuration. It should be obvious that the FORTRAN-C interfacing is a nontrivial step. Specifically, note how each machine defines the module name, as well as the argument types in some cases, differently. One must exercise a great deal of caution, when attempting to implement a port, to correctly predict how this step is to done. It is hoped that the examples provided in the form of the source files will give the necessary hints at where to start on such a venture. Often the appropriate symbols are defined automatically. To find out which one's are, just consult the compiler and pre-processor (`cpp`) documentation for your particular application. On each of the machines listed, the call is invoked via the uniform FORTRAN call:

```
CALL EXDATE( STRING )
```

## 5.1 User Interface Routines

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

### 5.1.1 Get Today's Date (EXDATE)

```
CALL EXDATE( STRING )
```

STRINGCHARACTER\last 8Write Only 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.

### 5.1.2 Get Time of Day (EXTIME)

CALL EXTIME( STRING )

STRINGCHARACTER\last 8Write Only 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.

### 5.1.3 Get Accumulated Processor Time (EXCPUS)

CALL EXCPUS( CPUSEC )

CPUSECREALWrite Only 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.

### 5.1.4 Get Operating Environment Parameters (EXPARM)

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

HARDCHARACTER\last 8Write Only System Hardware ID. For example, “CRAY-1/S”.

SOFTCHARACTER\last 8Write Only System Software ID. For example, “COS 1.11”.

MODEINTEGERWrite Only Job mode: 0 = batch, 1=interactive. For this purpose, an interactive environment means that the user can respond to unanticipated questions.

KCSUINTEGERWrite Only Number of character storage units per base system unit.

KNSUINTEGERWrite Only Number of numeric storage units per base system unit.

IDAUINTEGERWrite Only Units of storage which define the size of unformatted direct access I/O records: 0 = character, 1 = numeric. (For a more in-depth discussion of this topic, the reader is referred to the VAX FORTRAN manual, section 13.1.21.)

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 \* (KCSU + KNSU-1) ) / KCSU.

### 5.1.5 Get Unit File Name or Symbol Value (EXNAME)

CALL EXNAME( IUNIT, NAME, LN )

IUNITINTEGERRead Only Unit number if IUNIT > 0, or symbol ID if IUNIT ≤ 0.

NAMECHARACTER\last (\last )Write Only 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.

LNINTEGERWrite Only 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 pre-connection 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.

## 5.2 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 EXMEMORY (section [Ref: sec:exmemory] ) in conjunction with the memory manager is discouraged.

### 5.2.1 Convert String to Uppercase (EXUPCS)

CALL EXUPCS( STRING )

STRINGCHARACTER\last (\last )Read and Write Character string for which lowercase letters will be translated to uppercase. All other characters which are not in the printable ASCII character set are converted to spaces.

### 5.2.2 Prompt/Read/Echo Input Record (EXREAD)

CALL EXREAD( PROMPT, INPUT, IOSTAT )

PROMPTCHARACTER\last (\last )Read Only Prompt string.

INPUTCHARACTER\last (\last )Write Only Input record from standard input device.

IOSTATINTEGERWrite Only 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.

### 5.2.3 Evaluate Numeric Storage Location (IXLNUM)

NUMLOC = IXLNUM( NUMVAR )

NUMVARINTEGER or REALRead Only Any numeric variable.

NUMLOCINTEGERWrite Only Numeric location of NUMVAR. This value is an address measured in ANSI FORTRAN numeric storage units.

### 5.2.4 Evaluate Character Storage Location (IXLCHR)

CHRLOC = IXLCHR( CHRVAR )

CHRVARCHARACTERRead Only Any character variable.

CHRLOCINTEGERWrite Only Character location of CHRVAR. This value is an address measured in ANSI FORTRAN character storage units.

### 5.2.5 Get/Release Memory Block (EXMEMY)

CALL EXMEMY( MEMREQ, LOCBLK, MEMRTN )

MEMREQINTEGERRead Only Number of numeric storage units to allocate if MEMREQ > 0, or release if MEMREQ < 0.

LOCBLKINTEGERRead (release) or Write (allocate) 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.

MEMRTNINTEGERWrite Only 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.

## 5.3 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, though *not* functional. The skeleton routines can serve as

temporary placeholders for use when developing the Extension Library on a new system. Such an approach allows one to achieve interim support during the development period so that the functional version of each module can be developed individually, if necessary.

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.

### **5.3.1 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. IXLCHR returns unity.
10. 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.

## 6 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.

### 6.1 Free Field Input

The SUPES free field input system consists of four subroutines: FREFLD (section [Ref: sec:frefld] ), FFISTR (section [Ref: sec:ffistr] ), GETINP (section [Ref: sec:getinp] ), and STRIPB (section [Ref: sec:stripb] ). All of these routines are written in fully standard ANSI FORTRAN.

FREFLD calls the extension library routine EXUPCS (section [Ref: sec:exupcs] ). If only the skeleton version of EXUPCS is available, case insensitivity of input data (rule [Ref: itm:case] of section [Ref: sec:syntax] ) can not be guaranteed.

FFISTR is the input line parsing routine. It is called by FREFLD, but the user is free to call it independently. The input line may be of arbitrary length.

GETINP calls the extension library routine EXREAD (section [Ref: sec:exread] ). If only the skeleton version of EXREAD is available, GETINP will not prompt nor guarantee echo when reading from the standard input device ( $KIN = 0$ ).

#### 6.1.1 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 non-standard 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.

### 6.1.2 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. A sample session from a Sun 4/60 Workstation follows:

```
% ffrtest          <-- At the system prompt, enter the program name.
  1: This is an example <-- At the SUPES prompt, the user enters a line,
etc.
  NFIELD =      4
    I      KV(I)      CV(I)      RV(I)      IV(I)
    1      0  "THIS      "          0.          0
    2      2  "IS        "          0.          0
    3      0  "AN        "          0.          0
    4      0  "EXAMPLE   "          0.          0
    5     -1  "          "          0.          0
  2: Another line = example.
  NFIELD =      3
    I      KV(I)      CV(I)      RV(I)      IV(I)
    1      0  "ANOTHER   "          0.          0
    2      0  "LINE      "          0.          0
    3      0  "EXAMPLE.  "          0.          0
    4     -1  "          "          0.          0
    5     -1  "          "          0.          0
  3: This is a further 3.e5
  NFIELD =      6
    I      KV(I)      CV(I)      RV(I)      IV(I)
    1      0  "THIS      "          0.          0
    2      2  "IS        "          0.          0
    3      2  "A         "          0.          0
    4      0  "FURTHER   "          0.          0
    5      2  "3.E5      "          3.000E+05    300000
  4: exit
  NFIELD =      1
    I      KV(I)      CV(I)      RV(I)      IV(I)
    1      0  "EXIT      "          0.          0
```

```

2          -1  "          "          0.          0
3          -1  "          "          0.          0
4          -1  "          "          0.          0
5          -1  "          "          0.          0
5: ^C          <-- To exit, the user enters a ^C.
%
```

## 6.2 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.

### 6.2.1 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 converted to upper case then 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.

### 6.2.2 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.

### 6.2.3 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 [Ref: sec:skel] ).
2. Alter the memory manager subroutine MDINIT as follows:

ORIGINAL

```
DIMENSION MYV(1)
```

ALTERED

```
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.

### 6.2.4 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. While the proper implementation of the memory management test program requires an in-depth examination of the corresponding source file, a short test run on a Cray running the UNICOS operating system follows (comments are included after an arrow, <--):

```
% memtest      <-- At the system prompt, enter program name.
```

```

FUNC: mdinit      <-- At the SUPES prompt, the user enters a string, etc.
FUNC: mcinit
FUNC: mdwait
FUNC: mdrsrv real1 108
    POINTER:      -65733
FUNC: mcrsrv char 850
    POINTER:      -532696
FUNC: mdrsrv real2 108
    POINTER:      -65733
FUNC: mdexec
    POINTER BEFORE   -65733
    POINTER AFTER    17879  <-- Having the pointer updated is vital!
FUNC: mdlst
*****
0 * * * * * D I C T I O N A R Y * * * * *
0
    NUMERIC  CHARACTER
NAME      LOCATION      LENGTH      LENGTH
    1 CHAR          17664          107          850
    2 REAL1         17771          108          -1
    3 REAL2         17879          108          -1
0
    TOTAL                      323          850
0 * * * V O I D   T A B L E * * *
0
    LOCATION      LENGTH
    1              17987          61
0
    TOTAL                      61
*****
0 * * * * * O R D E R E D   L I S T * * * * *
0
    NUMERIC  CHARACTER
NAME      LOCATION      LENGTH      LENGTH
    1 CHAR          17664          107          850
    2 REAL1         17771          108          -1
    3 REAL2         17879          108          -1
    4              17987          61
    BLOCK SIZE                      384          850
    ALLOCATED TOTAL                  384          850
    GRAND TOTAL                      384          850
FUNC: exit
    STOP                          in MEMTEST
%
```

### 6.3 Extension Library Implementation

Implementing the SUPES 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.

As has been mentioned previously, this version represents a change in philosophy regarding the procedure for implementing a port of the extension library. Specifically, many of the features of the extension library require a richer data type than is available in

ANSI FORTRAN 77. For example, the requirement to do pointer assignment for the memory management made it desirable to utilize a more flexible programming language. The language chosen was C . A direct consequence of this is that the entire SUPES extension library is now coded in a single set of source files across all supported machines. Among the advantages are:

1. It reduces the amount of bookkeeping that is necessary to maintain the library across a number of machine architectures at a given site,
2. It now allows for a codified approach to building the library on any given machine, and finally,
3. It permits one to use the current source as an example for a future port.

Of course, these advantages do come at a cost. The FORTRAN–C interface must now be handled at the source level in the extension library. This is an extremely system dependent area. However, most systems do allow for such a scenario, and, accordingly, it tends to be documented quite extensively.

The code should be well commented and references to appropriate system manuals should be included.

The original FORTRAN version of the skeleton library will continue to be part of the distribution. To use them, it is recommended that one start with the skeleton library routines and gradually add system dependent code to provide full capability. Examples are provided in the distribution in the form of the original source code for VAX/VMS and Cray CTSS with the CFTLIB FORTRAN run-time library. In fact, for the latter case that code represents the only method of implementation of SUPES. This is entirely due to the lack of a suitable FORTRAN-C interfacing scheme under that system.

If this is the desired plan, 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 and/or C Run Time Library manuals for the particular operating system.
3. EXREAD, EXNAME, EXMEMY, and IXLCHR. These routines require a more intimate knowledge of the operating system. A substantial set of system documentation may be required to accomplish these tasks.

### **6.3.1 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 pre-connection. 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 lower case 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.

In all instances, the C programming language provides a clean method for returning the address for `IXLNUM`. In some cases it may be necessary to convert the address to numeric units. For example, addresses on VMS must be divided by four to convert from bytes to numeric storage units. The same cannot necessarily be said for a character address as returned by `IXLCHR`. The reader is referred to the source file `ixlchr.c` for further details on how to attack this problem.

`EXMEMORY` is the most crucial routine in the extension library—and one the primary reasons for choosing to do the extension library in C. As opposed to in the past, this latest approach has made it one of the most straightforward in the entire extension library. However, care should still be taken to ensure that both memory block locations and sizes are measured in numeric storage units. In the current version of SUPES, memory is allocated in blocks of 512 bytes (a number which can be changed at compile time) to improve performance. `EXMEMORY` should return the precise amount of memory allocated. Any memory that is given by the system, but not requested by the user is kept track of in a void table by the memory manager. So, it is generally unnecessary to keep track of memory blocks allocated via `EXMEMORY`.

### 6.3.2 Extension Library Test Program

A short program which exercises all features of the SUPES 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 `EXMEMORY`, should be developed as needed. An example session on a Sun 4/60 Workstation follows (with comments offset by an arrow, `<--`):

```
% setenv FOR001 junk.dat <-- Test EXNAME.
% exttest      <-- At the system prompt, invoke the procedure.
TST: ldkj     <-- At the SUPES prompt, the user enters a string.
Input line = LDKJ <-- The input line is returned in upper case.
Date = 12/18/89
Time = 09:58:05
Unit 1 name = junk.dat
Unit 10 name =
Symbol 1 =
Processor = Sun4      System = OS4.0.3c  Mode = 1
Character, Numeric, D/A Units: 4 1 0
Memory block location and length: 24700 128
Numeric difference = 4
Character difference = 4
CPU time = 7.00000E-02
```

## 6.4 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.

Any known bugs or idiosyncrasies.

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



## 7 References

1. American National Standard Programming Language FORTRAN, American National Standards Institute, Inc., ANSI X3.9-1978, New York, 1978.
2. D. P. Flanagan, W. C. Mills-Curran, and L. M. Taylor, "SUPES A Software Utilities Package for the Engineering Sciences," SAND86-0911, Sandia National Laboratories, Albuquerque, NM, September 1986.



## 8 SITE SUPPLEMENTS

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 (CFS) with those files having a file type of “.STX” being stored in Standard Text format. The SNLA installation of SUPES contains both the previous and new versions. This is done for two reasons, first, to provide the necessary compatibility during an interim migration period, and, second, to assure that current users of Cray/CTSS continue to have a point of reference for the SUPES library.

The previous version is stored under the root directory “/SUPES”. The table below documents the files stored in this directory.

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

The current version is stored under the CFS root directory “/SUPES2\_1” in the following files (the last two are *not* in Standard Text Format):

<i>Node</i>	<i>Contents</i>
<b>FRE_FLD.STX</b>	Free field reader source code
<b>MEM_MGR.STX</b>	Memory manager source code
<b>EXT_LIB.STX</b>	Portable C extension library source code
<b>FRR_TEST.STX</b>	Free field reader test program source code
<b>MEM_TEST.STX</b>	Memory manager test program source code
<b>EXT_TEST.STX</b>	Extension library test program source code
<b>SUPES2_1.BCK</b>	The version 2.1 distribution in VMS BACKUP format
<b>SUPES2_1.TAR</b>	The version 2.1 distribution in UN*X TAR format

[Footnote: Note that these files are *not*/ stored in Standard Text] The current extension library has been ported to run on the following machine/operating system combinations:

1. Sun 3 and Sun 4 running SunOS operating system version 4.0.3 and later,
2. VAXen running VMS version 4.5 and later,
3. Cray X/MP and Y/MP running UNICOS version 5.0 and later, and
4. Alliant F/X 8 running Concentrix 5.0.0.

A notable exception to the above list is the Cray using the CTSS operating system. This configuration still requires the FORTRAN source code for the extension library that was provided in previous implementations of SUPES. This code continues to be included in the current standard SUPES distribution, though a build procedure designed for this system is not.

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.

## **8.1 Site Supplement For 1500 VAX Cluster (VAX/VMS 5.1)**

### **8.1.1 Linking**

The SUPES package is accessed on the 1500 VAX CLUSTER (SAV01, SAV03, SAV07 and SAV08) as an object library located via either of two system logical names. Which one that the user uses depends on which version that he or she wants to use. The older SUPES routines are linked to an application program as follows:

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

While the newer version can be accessed at link time via:

```
$ LINK your_program,SUPES2_1/LIB,etc.
```

The last of the above commands assumes that the SUPES2\_1 library has been installed by someone using the VMSINSTALL.COM command procedure. If that is not the case, then the user will be informed by the LINKer that there is an abundance of unsatisfied external references that have been made. To avoid this scenario, one should be sure to provide the VAX C Run Time Library to the LINK command. One way to do this appropriately is to modify the above link command as:

```
$ LINK your_program,SUPES2_1/LIB,SYS$LIBRARY:VAXCRTL/LIB
```

The alternative is to define the logical LNK\$LIBRARY to be SYS\$LIBRARY:VAXCRTL. For systems which already have this logical assigned, define the logical LNK\$LIBRARY\_n, where n is the smallest integer for which the corresponding logical has not been assigned. (Hints about how to go about this are provided in the file [.BUILD]VMSINSTALL.COM.)

### 8.1.2 Defining Unit/File or Symbol/Value for EXNAME:

Both versions of SUPES use this extension library call in the same manner. 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) )
```

One caveat to note regarding the above sequence is that if the ASSIGN statement is not performed, the user program will abort with an error in the OPEN statement. A possible, or preferred code sequence is:

```
CALL EXNAME( 7, FILENM, LN )
IF( LN .EQ. 0 ) THEN      ! EXNAME returns a zero for LN if no ASSIGN
                          ! has been performed. Use the system default.
    OPEN( 7 )
ELSE                      ! I've found an ASSIGN'd filename, use it.
    OPEN( 7, FILE=FILENM )
ENDIF
```

where the system default mentioned in the above FORTRAN comment is a file named “FOR007.DAT” in the current default directory.

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 )
```

### 8.1.3 Interface to EXREAD

EXREAD will prompt to, and read from, SYS\$INPUT. It will automatically echo to SYS\$OUTPUT if that device is a terminal. However, if a program is run in a context where SYS\$OUTPUT is *not* a terminal, such as from within a command procedure, the input is not echoed—the user will have to control this himself with an appropriate call parameter to the routine FREFLD. 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.

### 8.1.4 Additional Comments Regarding SUPES2\_1

When attempting to redefine the logical SYS\$OUTPUT, the user should note that under VMS, the mixed language environment has a minor side effect: two versions of the output file are created by default. To avoid this scenario, he or she, will have to explicitly open the file. The following code segment demonstrates the required command sequence:

```
$ OPEN/WRITE SYSOUT OUTPUT.DAT
```

```
$ ASSIGN/USER_MODE SYSOUT SYS$OUTPUT
$ RUN PROG
$ CLOSE SYSOUT
```

Finally, the user should be aware that his or her program is being linked with the VAX C Run-time Library. Consequently, certain function, subroutine or more generally, external symbol names [Footnote: This does *not* include FORTRAN keywords, for example, READ and WRITE statements.] might be in conflict with some of these run time library functions. These include the names:

1. SPRINTF
2. GETENV
3. READ
4. WRITE
5. STRNCPY
6. STRCPY
7. STRLEN
8. SBRK
9. BRK
10. ISATTY
11. PERROR
12. ISASCII
13. ISCNTRL
14. ISALPHA
15. ISLOWER
16. TOUPPER

The remedy is to redefine any user-supplied conflict when warned by the linker of multiply defined symbol names.

### **8.1.5 Source Code**

The source code for the old 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. Conversely, the new version is in “/SUPES2\_1/EXT\_LIB.STX”.

## **8.2 Site Supplement for SNLA CRAY-1/S (COS 1.11)**

### **8.2.1 Linking**

The newer version of SUPES is not available for this system. However, the older package can still be accessed on the SNLA CRAY-1/S using COS 1.11 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.
```

### 8.2.2 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) )
```

Again, the more suitable code sequence is

```
CALL EXNAME( 7, FILENM, LN )
IF( LN .EQ. 0 ) THEN      ! EXNAME returns a zero for LN if no ASSIGN
                          ! has been performed. Use the system default.
    OPEN( 7 )
ELSE                      ! I've found an ASSIGN'd filename, use it.
    OPEN( 7, FILE=FILENM )
ENDIF
```

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—this is also the system default.

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 )
```

### 8.2.3 Interface to EXREAD

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

### 8.2.4 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'.

### 8.2.5 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.

## 8.3 Site Supplement for SNLA CRAY-1/S (UNICOS)

### 8.3.1 Linking

The new version of SUPES is the only one that is available for this system. It resides in the directory `/usr/local/lib` with the file name `libsupes.a`

In what follows, an example of how the SUPES routines can be linked to an application program is given:

```
% cf77 -o your-executable your-source.f -lsupes.
```

### 8.3.2 Defining Unit/File or Symbol/Value for EXNAME

A file name is connected to a unit number via an environment variable of the form FOR0nn, where “nn” is a two digit integer indicating the FORTRAN unit number. For example, if the user is currently running under the shell program `/bin/csh`, the required sequence is:

```
% setenv FOR007 cards.dat
```

This causes the following FORTRAN statements to open 'cards.dat' on unit 7.

```
CALL EXNAME( 7, FILENM, LN )
IF( LN .EQ. 0 ) THEN          ! EXNAME returns a zero for LN if no ASSIGN
                              ! has been performed. Use the system default.
    OPEN( 7 )
ELSE                          ! I've found an ASSIGN'd filename, use it.
    OPEN( 7, FILE=FILENM )
ENDIF
```

From the Bourne Shell, `/bin/sh`, the following sequence is required:

```
$ FOR007=cards.dat
$ export TERM
```

If no file has been assigned, a system default file name of the form `fort.nn`, where “nn” is a one (if less than ten) or two digit integer indicating the FORTRAN unit number that will be written.

Similarly, EXNAME looks for an environment variable of the form EXTnn. So that

```
% setenv EXT05 hello
```

will cause the following call to return NAME="hello" and LN=5.

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

### 8.3.3 Interface to EXREAD

EXREAD will read from `stdin` and automatically echo to `stdout`.

### 8.3.4 Known Problems

The Cray running UNICOS appears to have some compiler specific problems when linking programs of differing levels of optimization. To alleviate this situation, two versions of the SUPES library are maintained in `/usr/local/lib`, `libsupes.a` and `lib-supesnopt.a`. The user will be responsible for linking to the appropriate library.

### 8.3.5 Source Code

The source code for the extension library for the UNICOS operating system is stored in the SNLA Central File System under node `"/SUPES2_1/EXT_LIB.STX"` in SNLA Standard Text format.

## 8.4 Site Supplement For SNLA CRAY X-MP/24 (CTSS/CFTLIB 1.11 or 1.14)

### 8.4.1 Linking

The old SUPES package is all that is currently available on this system. It 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.

### 8.4.2 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 )
```

### 8.4.3 Interface to EXREAD

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
```

### 8.4.4 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=\last 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.

### 8.4.5 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.

## 8.5 Site Supplement for SNLA Alliant FX/8 (Concentrix 5.0.0)

### 8.5.1 Linking

The new version of SUPES is the only one that is available for this system. It resides in the directory `/usr/local/lib` with the file name `libsupes.a`

In what follows, an example of how the SUPES routines can be linked to an application program is given:

```
% fortran -o your-executable your-source.f -lsupes.
```

### 8.5.2 Defining Unit/File or Symbol/Value for EXNAME

A file name is connected to a unit number via an environment variable of the form FOR0nn, where “nn” is a two digit integer indicating the FORTRAN unit number. For example, if the user is currently running under the shell program `/bin/csh`, the required sequence is:

```
% setenv FOR007 cards.dat
```

This causes the following FORTRAN statements to open ‘`cards.dat`’ on unit 7.

```
CALL EXNAME( 7, FILENM, LN )
IF( LN .EQ. 0 ) THEN          ! EXNAME returns a zero for LN if no ASSIGN
                              ! has been performed. Use the system default.
    OPEN( 7 )
ELSE                          ! I've found an ASSIGN'd filename, use it.
    OPEN( 7, FILE=FILENM )
ENDIF
```

From the Bourne Shell, `/bin/sh`, the following sequence is required:

```
$ FOR007=cards.dat
$ export TERM
```

If no file has been assigned, a system default file name of the form `fort.nn`, where “nn” is a one (if less than ten) or two digit integer indicating the FORTRAN unit number that will be written.

Similarly, EXNAME looks for an environment variable of the form EXTnn. So that

```
% setenv EXT05 hello
```

will cause the following call to return NAME=“hello” and LN=5.

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

### 8.5.3 Interface to EXREAD

EXREAD will read from `stdin` and automatically echo to `stdout`.

### 8.5.4 Source Code

The source code for the extension library for the Alliant is stored in the SNLA Central File System under node `"/SUPES2_1/EXT_LIB.STX"` in SNLA Standard Text format.

## 8.6 Site Supplement for SNLA Sun Workstations (SunOS version 4)

### 8.6.1 Linking

The new version of SUPES is also the only one that is available for this system. Note that the SUPES installation must have been performed according to the installation instructions [Ref: sec:install]. If so, then it resides in the directory `/usr/local/lib` with the file name `libsupes.a`

In what follows, an example of how the SUPES routines can be linked to an application program is given:

```
% f77 -o your-executable your-source.f -lsupes.
```

### 8.6.2 Defining Unit/File or Symbol/Value for EXNAME

A file name is connected to a unit number via an environment variable of the form `FOR0nn`, where `"nn"` is a two digit integer indicating the FORTRAN unit number. For example, if the user is currently running under the shell program `/bin/csh`, the required sequence is:

```
% setenv FOR007 cards.dat
```

This causes the following FORTRAN statements to open `'cards.dat'` on unit 7.

```
CALL EXNAME( 7, FILENM, LN )
IF( LN .EQ. 0 ) THEN          ! EXNAME returns a zero for LN if no ASSIGN
                              ! has been performed. Use the system default.
    OPEN( 7 )
ELSE                          ! I've found an ASSIGN'd filename, use it.
    OPEN( 7, FILE=FILENM )
ENDIF
```

From the Bourne Shell, `/bin/sh`, the following sequence is required:

```
$ FOR007=cards.dat
$ export TERM
```

If no file has been assigned, a system default file name of the form `fort.nn`, where `"nn"` is a one (if less than ten) or two digit integer indicating the FORTRAN unit number that will be written.

Similarly, EXNAME looks for an environment variable of the form `EXTnn`. So that

```
% setenv EXT05 hello
```

will cause the following call to return NAME="hello" and LN=5.

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

### **8.6.3 Interface to EXREAD**

EXREAD will read from `stdin` and automatically echo to `stdout`.

### **8.6.4 Source Code**

The source code for the extension library for the Sun is stored in the SNLA Central File System under node "/SUPES2\_1/EXT\_LIB.STX" in SNLA Standard Text format.