

AD-A093 262

VIRGINIA POLYTECHNIC INST AND STATE UNIV WASHINGTON --ETC F/6 9/2

LBSIM: AN EXTENSION OF THE SIMULA: (TRADEMARK) LIBRARY.(U)

AUG 80 R J ORGASS

AFOSR-79-0021

UNCLASSIFIED

VPI/SU-TM-80-4

AFOSR-TR-80-1290

NL

1 of 1
AD
AFOSR-79-0021



END
DATE
FILMED
281
DTIC



AFOSR-TR. 80 - 1290

12

VIRGINIA POLYTECHNIC INSTITUTE AND STATE UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE
GRADUATE PROGRAM IN NORTHERN VIRGINIA

P. O. Box 17186
Washington, D. C. 20041
(703) 471-4600

LEVEL II

LIBSIM: AN EXTENSION OF THE SIMULA* LIBRARY†

Richard J. Orgass

Technical Memorandum No. 80-4

August 28, 1980

STIC
DEC 23 1980
F

ABSTRACT

The DEC-10 SIMULA library contains a number of procedures that are not available in the IBM SIMULA library. A subset of these procedures as well as approximations to procedures in class SAFEIO (distributed with DEC-10 SIMULA) have been implemented for use with IBM SIMULA and collected in a TXTLIB. This report describes these procedures and gives directions for accessing and using the procedures.

* SIMULA is a registered trademark of the Norwegian Computing Center.

† Research sponsored by the Air Force Office of Scientific Research, Air Force Systems Command, under Grant No. AFOSR-79-0021. The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation hereon.

Approved for public release;
distribution unlimited.

DDC FILE COPY.

80 12 22 197

Located at Dulles International Airport—400 West Service Road

Copyright, 1980

by

Richard J. Orgass

General permission to republish, but not for profit, all or part of this report is granted, provided that the copyright notice is given and that reference is made to the publication (Technical Memorandum No. 80-4, Department of Computer Science, Graduate Program in Northern Virginia, Virginia Polytechnic Institute and State University), to its date of issue and to the fact that reprinting privileges were granted by the author.

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER 18 AFOSR-TR-80-1290	2. GOVT ACCESSION NO. DAAG 3262	3. RECIPIENT'S CATALOG NUMBER 14	
4. TITLE (and Subtitle) LIBSIM: AN EXTENSION OF THE SIMULA LIBRARY		5. TYPE OF REPORT & PERIOD COVERED VPI/SU-TM-80-4	
6. PERFORMING ORG. REPORT NUMBER		7. AUTHOR(s) Richard J. Orgass	
8. CONTRACT OR GRANT NUMBER(s)		9. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304/A5	
10. CONTROLLING OFFICE NAME AND ADDRESS Dept. of Computer Science Virginia Polytech. Inst. and State University		11. REPORT DATE 28 August 1980	
12. CONTROLLING OFFICE NAME AND ADDRESS Air Force Office of Scientific Research/NM Boiling AFB, Washington, D.C. 20332		13. NUMBER OF PAGES 15	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) T. ...		15. SECURITY CLASS. (of this report) UNCLASSIFIED	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release, distribution unlimited.			
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		18. DISTRIBUTION/Availability Codes IS GRA&I MIC TAB Unannounced Justification	
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		20. ABSTRACT (Continue on reverse side if necessary and identify by block number) The DEC-10 SIMULA library contains a number of procedures that are not available in the IBM SIMULA library. A subset of these procedures as well as approximations to procedures in class SAFEIO (distributed with DEC-10 SIMULA) have been implemented for use with IBM SIMULA and collected in a TXLIB. This report describes these procedures and gives directions for accessing and using the procedures.	

411 731

LIBSIM: AN EXTENSION OF THE SIMULA LIBRARY

1. Introduction

In the course of bringing a number of DEC-10 SIMULA interactive programs into satisfactory operation in the VM/CMS environment, a collection of generally useful SIMULA procedures has been developed and placed in a CMS TXTLIB for general use. This report provides documentation for the procedures in this library.

A set of three procedures that considerably simplify the creation of terminal dialogs is described in Section 2.

When doing any amount of text processing in SIMULA it is very convenient to have a number of procedures that are not in the standard IBM SIMULA library available. Most of the text processing procedures described in Section 3 are CMS implementations of procedures that are part of the DEC-10 SIMULA library.

Sorting procedures for one dimensional arrays of types integer, real and character are described in Section 4. These sort procedures are a structured implementation of Floyd's tree-sort_3.

A variety of utility procedures that serve a specific function are described in Section 5. They include a means for entering CMS subset, a reasonable procedure to terminate execution without a GO TO directed to the end of the program and dummy procedures to retain calls to the DEC-10 SIMULA procedures freeze and enter-debug.

The character set conversion procedures described in Section 6 provide the resources to support input from both bit-paired and key-paired ASCII/APL terminals and to provide translations between the ASCII and EBCDIC character sets.

There are other procedures in LIBSIM which support the procedures described here. Since these procedures are not intended for general use, these procedures are not mentioned in this report. The source files for these procedures contain the documentation needed for maintenance.

In the body of this report, procedures are described after exhibiting the heading of the procedure declaration. This declaration should not be used in a running program. The procedures should simply be declared as external procedures. For example, in Section 3 there is a description of a procedure conc2 and the heading:

```
TEXT PROCEDURE conc2(t1, t2);  
    VALUE t1, t2; TEXT t1, t2;
```

is exhibited. In a program that uses this procedure, it should be declared as:

```
EXTERNAL TEXT PROCEDURE conc2;
```

Unless there is some statement in the body of this report to the contrary, all procedures described here should be declared in this way.

[In this report, procedure names are given as an identifier that is long enough to be a meaningful name for the procedure. The SIMULA compiler automatically truncates all external procedure identifiers to the first seven characters. Therefore, entries in LIBSIM TXTLIB are truncated to the first seven characters. Users are encouraged to use the full procedure name in their programs because the txtlib will be modified if the SIMULA compiler no longer truncates external procedure identifiers.]

The txtlib LIBSIM is on the 191 disk of userid CSDULLES. If you are planning to use these procedures in your programs, you must link to this disk (read password all) as a read only extension of your A disk. In addition, your global statement should be:

```
global txtlib libsim mvslib
```

or

```
global txtlib libsim simlib
```

This global must be executed after you link to the 191 disk of CSDULLES.

The source files for all of the procedures in libsim are in the genlib LIBSIM on the 192 disk of CSDULLES (read password all).

It has been determined that there are some differences between the SIMULA run time library for MVS SIMULA and for CMS SIMULA. The system default simlib is the CMS SIMULA library. There are some programs that will execute correctly in CMS only if they use the MVS SIMULA run time library. The MVS SIMULA library is on the 191 disk of CSDULLES as file MVSLIB TXTLIB.

A reasonable set of commands to execute before compiling and executing programs that use the procedures described here is:

```
cp link csdulles 191 330 rr all  
access 330 b/a  
global txtlib libsim mvslib
```

or:

```
cp link csdulses 191 330 rr all
access 330 b/a
global txtlib libsim simlib
```

2. Terminal Dialog Procedures

The three procedures described in this section have the following properties. When the procedure is called the parameters include the question to be asked, a default answer, and a procedure to print a further explanation of the question on the terminal. The return value of the procedure is the user's response. If the user responds with a carriage return, the return value is the default value and if the user responds with a question mark (?) the procedure to explain the question is invoked and then the question is asked again.

For example, suppose a program wishes to read an input file name from the terminal into a text variable `file_spec`. This can be accomplished by executing the following statement:

```
file_spec :- textrequest("Input file: ", NOTEXT, TRUE);
```

The first parameter is the question that is to be printed on the terminal. The second parameter is the default answer and since this parameter is `NOTEXT`, there is no default answer. The third parameter is the constant `TRUE` to indicate that no help is available. When the statement is executed, the terminal transcript will look like this:

```
Input file: ?
No help available.
Input file:
Default value may not be selected. Please try again.
Input file: letter simula
```

After this, the return value of `textrequest` is the text "letter simula". In the first response, the user asked for an explanation of the query by responding with a "?". Since the call to `textrequest` did not include a help procedure, the appropriate message was printed. Next, the second response was an empty line indicating that the default value was desired. Since no default value was specified in the procedure call, a corrective response was printed and then the question was asked again. Finally, the third response was a character string and this string became the return value of the procedure.

As a second example, suppose that the statement:

```
f :- textrequest("Output file: ",
                 "letter data",
                 TRUE);
```

is executed. In this case the terminal transcript would look like this:

```
Output file: /letter data/: ?  
No help available.  
Output file: /letter data/:
```

Since the second response was simply a carriage return, the return value of `textrequest` is the string "letter data". On the other hand, if the response were some other character string, then this string would be the return value.

Suppose it is desirable to provide a help message in response to the input "?". This might be accomplished by writing the following procedure:

```
BOOLEAN PROCEDURE help;  
BEGIN  
    Outimage;  
    Outtext("This file will contain ");  
    Outtext("a list of addresses");  
    Outimage;  
    Outtext("after the program is executed.");  
    Outimage  
END of help;
```

Executing the statement

```
f :- textrequest("Output file: ",  
                 "letter data",  
                 help);
```

might generate the following transcript:

```
Output file: /letter data/: ?  
This file will contain a list of addresses  
after the program is executed.  
Output file: /letter data/: mylib address
```

After this transcript, the statement concludes by assigning the text object "mylib address" to `f`.

The heading of the declaration of `textrequest` is:


```

TEXT PROCEDURE textrequest(prompt, default, no_help);
    NAME prompt, default, no_help;
    VALUE default;
    TEXT prompt, default;
    BOOLEAN no_help;

```

The formal parameter prompt is the question to be printed on the terminal and the formal parameter default is the value to be returned if the user's response is a carriage return. The parameter no_help is to be TRUE if there is no help available for this query. If there is help available for this query, it is printed by a boolean procedure that returns the value FALSE.

The procedure booleanrequest is used to ask yes or no questions in very much the same way. The heading of the declaration of this procedure is:

```

BOOLEAN PROCEDURE booleanrequest(prompt,
                                   default,
                                   no_help);
    NAME prompt, no_help;
    VALUE default;
    TEXT prompt;
    BOOLEAN default, no_help;

```

The formal parameter prompt is the question that is to be printed on the terminal and the parameter default (which must be a boolean variable or constant) is the return value if the user responds by entering a carriage return. If the user responds with a "?" and if no_help is TRUE then the message "No help available." is printed on the terminal. On the other hand, if no_help is a boolean procedure that returns FALSE and prints an explanation, this text is printed instead of "No help available."

For example, if the procedure help6 is declared as:

```

BOOLEAN PROCEDURE help6;
BEGIN
    Outtext("If tabs may be used at indentation ");
    Outtext("answer "y",");
    Outimage;
    Outtext("otherwise answer "n".");
    Outimage;
END of help6;

```

Then the execution of the statement:

```

tabs := booleanrequest("Tabs in indentation: ",
                        FALSE,
                        help6);

```

initiates the following transcript:

```

Tabs in indentation: /n/: ?
If tabs may be used at indentation answer "y",
otherwise answer "n".
Tabs in indentation: /n/: why
Please answer y or n.
Tabs in indentation: /n/: y

```

After this transcript, the return value of `booleanrequest` is TRUE. If the last response had been an empty line or "n", then the return value would have been FALSE.

Note that the response must be one of the letters "y" or "n" either in upper or lower case.

The procedure `integerrequest` is used to prompt for an integer response and to check if the response is an integer within an acceptable range. The heading of this procedure is:

```

INTEGER PROCEDURE integerrequest(prompt,
                                default,
                                min,
                                max,
                                no_help);

    NAME prompt, no_help;
    VALUE default, min, max;
    TEXT prompt;
    INTEGER default, max, min;
    BOOLEAN no_help;

```

The formal parameter `prompt` is the question to be printed on the terminal and the formal parameter `default` is the value that is returned if the user responds by entering carriage return. After a response is read from the terminal, it is checked to confirm that it is an integer between `min` and `max` (inclusive). If the response fails this test, a corrective message is printed and the user is asked for a correct response. [The default value is also checked against the range if the user responds with a carriage return.] The formal parameter `no_help` is used to deal with the user response " " as described above.

If any integer is an acceptable response, the SIMULA defined constant `Maxint` may be used in a call. For example, if the default answer to the prompt "Enter any integer: " is 0, one might declare

```
EXTERNAL ASSEMBLY PROCEDURE Maxint;
```

and then execute the statement

```

result := integerrequest("Enter any integer: ",
                          0,
                          -Maxint,
                          Maxint,
                          TRUE);

```

As a more detailed example, consider the execution of the statement:

```
result := integerrequest("reserved words: ",
                          1,
                          0,
                          3,
                          TRUE);
```

The terminal transcript might look like this:

```
Reserved words: /0/: ?
No help available.
Reserved words: /0/: bye
The input was not an integer. Please try again.
Reserved words: /0/: 12
The input integer was out of the acceptable range [0,3].
Please try again.
Reserved words: /0/: 2
```

After this sequence of events, the return value of integerrequest is 2.

These three procedures provide most of the terminal prompting activities that are needed in many interactive programs. While it might be desirable to have a procedure to prompt for floating point numbers, the authors have not felt this need and, therefore, did not include it in the library.

Finally, a reminder: In Section 1 it was pointed out that all procedures described in this report are available as external procedures and are to be so declared. In the above discussion, the headings were exhibited to convey information about the procedures. In a program that uses these procedures, they should be declared as follows:

```
EXTERNAL TEXT PROCEDURE textrequest;
EXTERNAL BOOLEAN PROCEDURE booleanrequest;
EXTERNAL INTEGER PROCEDURE integerrequest;
```

A similar statement applies to all the procedures described in this report.

The procedures described in this section are adaptations of procedures in class SAFEIO as distributed with DEC-10 SIMULA.

3. Text Processing Procedures

A variety of procedures that substantially simplify writing elementary text processing procedures in SIMULA are described in this section. Most of the procedures share the property that they accept one or more text objects as parameters and that they

return a new text object which is different from all of the parameters. As in Section 2, the headings of the procedure declarations precede the description of the procedure. All of the procedures described here except for conc are to be declared as external text procedures.

```
TEXT PROCEDURE conc2(t1,t2);  
    VALUE t1, t2;  
    TEXT t1, t2;
```

The return value of this procedure is a new text object that is the concatenation of the text objects t1 and t2. The length of the return value is t1.length + t2.length. The attribute Pos of the return value is 1.

```
TEXT PROCEDURE exptabs(t);  
    VALUE t; TEXT t;
```

This procedure returns a text object which is the untabulated version of its text parameter. That is, tab characters (HT) are replaced with spaces under the assumption that tabs are set every eight spaces as per the ISO standard. The return value of this procedure will print on a terminal without hardware tabs in the same way that its parameter will print on a terminal with hardware tabs.

```
TEXT PROCEDURE frontstrip(t); TEXT t;
```

The return value of this procedure is a copy of the parameter except that leading blanks have been removed. For example, frontstrip(Image.Strip) is Image with both leading and trailing blanks removed. The procedure is a SIMULA coded version of a procedure with the same name in the DEC-10 SIMULA library.

```
INTEGER PROCEDURE hash(t,n);  
    VALUE t;  
    TEXT t;  
    INTEGER n;
```

This procedure returns an integer in the range [0,n-1] which is a reasonable hash code for the text object that is the first parameter. The hash coding has been found to be reasonably effective in a variety of applications. Readers who are interested in studying the algorithm in detail are referred to the source listing in LIBSIM GENLIB on the 192 disk of CSDULLES.

```
BOOLEAN PROCEDURE isinteger(t);  
    NAME t; TEXT t;
```

This procedure returns TRUE if its text parameter contains exactly one integer and FALSE otherwise.

```
TEXT PROCEDURE rest(t); TEXT t;
```

This procedure returns the portion of the text *t* that begins at *t.Pos* and ends at *t.Length*. That is, *rest(x)* is best viewed as an abbreviation for:

```
x.Sub(x.Pos,x.Length-x.Pos+1)
```

The procedure is taken from the DEC-10 SIMULA manual and is available in the library of DEC-10 SIMULA.

```
TEXT PROCEDURE upcase(t);  
  VALUE t; TEXT t;
```

The return value of this procedure is a new text object that is the same as the actual parameter except that lower case letters have been mapped into upper case letters. It is quite useful when reading input from the terminal in mixed case. The user response can be converted to upper case before further processing and this simplifies programs.

4. Sorting Procedures

R. W. Floyd described a sorting algorithm called Treesort 3 in Volume 7 (1964) of the Communications of the ACM (page 701). This $O(n \log n)$ sorting algorithm has been the subject of intensive study and variations of the algorithm have been verified both informally and formally.

The three sorting procedures described in this section are all trivial modifications of an exceptionally clear version of Floyd's algorithm. The only difference between the procedures is the type of the first parameter.

```
PROCEDURE csort(x, n);  
  NAME x; VALUE n;  
  CHARACTER ARRAY x;  
  INTEGER n;
```

This procedure sorts the array *x* from *x[1]* to *x[n]* into ascending order in place. The first parameter is modified to reflect the result of the sort.

```

PROCEDURE isort(x, n);
    NAME x; VALUE n;
    INTEGER ARRAY x;
    INTEGER n;

```

This procedure has the same properties as csort except that an integer array is sorted.

```

PROCEDURE rsort(x, n);
    NAME x; VALUE n;
    REAL ARRAY x;
    INTEGER n;

```

This procedure has the same properties as csort except that a real array is sorted.

5. Utility Procedures

The procedures described in this section provide communication with the operating system and have been found useful in a number of applications.

```

PROCEDURE abort(message);
    VALUE message; TEXT message;

```

This procedure prints the actual parameter on the terminal and then terminates execution. However, just before execution is terminated, the user is given the option of printing a symbolic dump on the terminal. Unlike the IBM SIMULA system procedure Error, there is no limit on the length of the formal parameter and the symbolic dump is optional. The procedure is an exact replacement for the DEC-10 SIMULA library procedure abort in IBM SIMULA.

```

PROCEDURE cmssubset;

```

When this procedure is called, the message "[Entering CMS sub-set.]" is printed on the terminal and the character sharp (#) is used to prompt for input lines. Each input line is interpreted as a CMS command and the system response is printed on the terminal. The set of commands that can be executed in this mode is the same as the set of commands that can be executed in the CMS system editor in CMS mode. Control is returned to the calling program when the terminal input line is the text "return" in any mixture of upper and lower case letters.

```

PROCEDURE enterdebug(maycontinue);
    BOOLEAN maycontinue;

```

This procedure is designed as a dummy for the DEC-10 SIMULA library procedure with the same name. In the DEC-10 environment, this procedure invokes SIMDDT which is not currently available in

IBM SIMULA. In the IBM SIMULA environment of SIMULA Version 7.00, the user is given the option of printing a symbolic dump on the terminal as a substitute for SIMDDT.

If the actual parameter of this procedure is TRUE, then execution of the program continues after the call to enterdebug. On the other hand, if the actual parameter is false, then execution is terminated in enterdebug (by means of a call on abort). Programs that use this procedure should print some sort of explanatory message on the terminal before calling enterdebug.

```
PROCEDURE freeze(returncode);  
    NAME returncode; INTEGER returncode;
```

This procedure is designed as a dummy for the DEC-10 SIMULA library procedure freeze in IBM SIMULA. The present version simply prints a message to the effect that freeze is not available and continues execution. Future versions of libsim may include a working version of the procedure.

A working version of this procedure would have the following properties: A module file is written to the first accessible disk in the user's search list. This module has the property that execution begins at the statement following the call to freeze. When execution continues, the value of the actual parameter indicates if the module is suitable for further execution. If the return code is 0, execution may continue. Other non-zero return codes will signal errors.

In DEC-10 SIMULA, the library procedure with this name performs the function described above.

6. Character Set Conversions

The procedures described in this section are primarily of interest to readers who are writing programs that are designed to support ASCII/APL terminals. However, the procedures ascii and ebcdic provide character set translation between ASCII and EBCDIC and may be useful in a variety of other applications.

```
EXTERNAL FORTRAN INTEGER PROCEDURE ascii;
```

The actual parameter of this procedure is an integer in the range [0,255] and the return value is an integer in the range [0,127]. The actual parameter is interpreted as an EBCDIC character code and the return value is the ASCII code for the same character subject to the assumptions described below.

It is assumed that the only characters of interest are the 128 characters that can be entered on an ASCII terminal and the EBCDIC characters that do not correspond to one of these characters are treated as nulls (ASCII 0, EBCDIC 0).

This procedure uses the ASCII-EBCDIC translation table that is used with the Memorex 1380 in the Virginia Tech VM system.

The multiple EBCDIC codes for certain characters, e.g., ~, |, {, }, are acknowledged and all versions are mapped into the corresponding ASCII character.

EXTERNAL FORTRAN PROCEDURE bpair;

This procedure has four parameters ichar, i1, i2, i3. The first parameter is a value parameter and the remaining parameters are name parameters.

This procedure is designed to be used when writing EBCDIC text to a bit-paired ASCII/APL terminal when the terminal is in the APL character set. In this mode, the terminal has upper case letters and lacks some of the graphics in the ASCII character set. In order to preserve, as best possible, the appearance of text written in the ASCII subset of EBCDIC, character translation is desirable. In particular, the following translation is performed:

- (1) Lower case letters are translated into upper case (APL) letters.
- (2) Upper case letters are translated into underlined upper case letters.
- (3) ASCII graphics are mapped into the corresponding graphic in the APL character set. The exceptions are as follows: The sharp (#) is mapped into the right tack. The percent (%) is mapped into diamond. The ampersand (&) is mapped into the APL and sign. The at sign (@) is mapped into the APL character alpha. The "hat" (^) is mapped into the APL high minus.

When the procedure is called, the first parameter should be the EBCDIC code for the character that is to be printed on the terminal. Upon procedure return, the values of i1, i2 and i3 can be used to print the corresponding APL character as follows:

```
Outchar(Char(i1));  
IF i2 NE 0  
  THEN BEGIN  
    Outchar(Char(i2));  
    Outchar(Char(i3))  
  END;
```

In this code fragment, it is assumed that the terminal is already in the APL character set.


```

TEXT PROCEDURE convtoapl(t,t_type);
    VALUE t, t_type;
    TEXT t;
    INTEGER t_type;

```

In accord with the value of the variable `t_type`, an EBCDIC text object is translated into a text object that will print in approximately the same form on an ASCII, key-paired APL or bit-paired APL terminal.

The translation that is performed is specified by the value of the parameter `t_type` as follows:

- (1) If `t_type = 0`, the actual parameter is returned as the value of the procedure.
- (2) If `t_type = 1`, then the actual parameter is modified so that it will print on a key-paired ASCII/APL terminal. The character translation that is performed is described with the procedure `kpair` above.
- (3) If `t_type = 2`, then the actual parameter is modified so that it will print on a bit-paired ASCII/APL terminal. The character translation that is performed is described with the procedure `kpair` above.

```

EXTERNAL FORTRAN INTEGER PROCEDURE ebcdic;

```

The single parameter of this procedure is a value integer. This integer, which must be in the range `[0,127]`, is interpreted as an ASCII character code and the return value of this procedure is the EBCDIC code of the same character and is in the range `[0,255]`.

Character translation is performed using the translate table used with the Memorex 1380 in the Virginia Tech VM system.

```

EXTERNAL FORTRAN PROCEDURE kpair;

```

This procedure has the same specifications as the procedure `bpair` described above with one minor modification: The translation is performed for key-paired ASCII/APL terminals.

```

CHARACTER PROCEDURE kpchar(n); REAL n;

```

The range of the parameter `n` of this procedure is the reals that are equal to the integers in `[0,127]`. Such a real is interpreted as the ASCII code of an APL character on a key-paired APL terminal. The return value of this procedure is the EBCDIC character that is to be sent to the terminal to print the character that corresponds to the parameter under this mapping. This procedure is the inverse of `kpcode` described below.

REAL PROCEDURE kpcod(c); CHARACTER c;

The range of this procedure is the ASCII subset of EBCDIC. The parameter is interpreted as an APL character typed on a key-paired ASCII/APL terminal and the return value is the real that is equal to the ASCII code for this APL character. This procedure is the inverse of kpchar described above.

7. Maintenance Information

The source files for the procedures in LIBSIM are contained in file LIBSIM GENLIB on the 192 disk of userid CSDULLES (read password ALL). A listing of the directory of the GENLIB follows.

ABORT	SIMULA	Al	80	F	2	21	03/08/80	12:37:30
BOOLEAN	SIMULA	Al	80	F	24	114	03/08/80	12:37:34
CMSSUBS	SIMULA	Al	80	F	139	63	03/08/80	12:37:37
CONC2	SIMULA	Al	80	F	203	9	03/08/80	12:37:41
CONVTOA	SIMULA	Al	80	F	213	114	03/08/80	12:37:44
CSORT	SIMULA	Al	80	F	328	79	03/08/80	12:37:47
ENTERDE	SIMULA	Al	80	F	408	42	03/08/80	12:37:52
EXPTABS	SIMULA	Al	80	F	451	42	03/08/80	12:37:56
FDHELP	SIMULA	Al	80	F	494	47	03/08/80	12:38:02
FILEDEF	SIMULA	Al	80	F	542	70	03/08/80	12:38:04
FREEZE	SIMULA	Al	80	F	613	9	03/08/80	12:38:07
FRONTST	SIMULA	Al	80	F	623	19	03/08/80	12:38:10
GETDDIN	SIMULA	Al	80	F	643	220	03/08/80	12:38:13
GETDDOU	SIMULA	Al	80	F	864	139	03/08/80	12:38:18
HASH	SIMULA	Al	80	F	1004	31	03/08/80	12:38:27
INITIAL	SIMULA	Al	80	F	1036	15	03/08/80	12:38:32
INTEGER	SIMULA	Al	80	F	1052	161	03/08/80	12:38:38
ISINTEG	SIMULA	Al	80	F	1214	39	03/08/80	12:38:47
ISORT	SIMULA	Al	80	F	1254	79	03/08/80	12:38:52
KPCHAR	SIMULA	Al	80	F	1334	26	03/08/80	12:38:59
KPCODE	SIMULA	Al	80	F	1361	26	03/08/80	12:39:05
PART	SIMULA	Al	80	F	1388	72	03/08/80	12:39:10
REST	SIMULA	Al	80	F	1461	8	03/08/80	12:39:15
RESTORE	SIMULA	Al	80	F	1470	14	03/08/80	12:39:22
RSORT	SIMULA	Al	80	F	1485	79	03/08/80	12:39:26
SIMXXX	SIMULA	Al	80	F	1565	17	03/08/80	12:39:31
UPCASE	SIMULA	Al	80	F	1583	39	03/08/80	12:39:46
XLATE	SIMULA	Al	80	F	1623	16	03/08/80	12:39:55
DDCNT	FORTTRAN	Al	80	F	1640	6	03/08/80	12:40:03
ASCII	RATFOR	Al	80	F	1647	148	03/08/80	12:40:15
BPAIR	RATFOR	Al	80	F	1796	271	03/08/80	12:40:29
EBCDIC	RATFOR	Al	80	F	2068	74	03/08/80	12:40:40
KPAIR	RATFOR	Al	80	F	2143	271	03/08/80	12:40:53
TABEXP	ASSEMBLE	Al	80	F	2415	251	03/08/80	12:41:06
VPIABORT	ASSEMBLE	Al	80	F	2667	33	03/08/80	12:41:21
VPISAVE	ASSEMBLE	Al	80	F	2701	167	03/08/80	12:42:35
VPIREST	ASSEMBLE	Al	80	F	2869	169	03/24/80	16:28:55
VPIINI	ASSEMBLE	Al	80	F	3039	305	03/25/80	12:21:21

VPIMOI	ASSEMBLE	A1	80	F	3345	229	04/16/80	14:04:25
TEXTREQ	SIMULA	A1	80	F	3575	116	04/22/80	12:57:38