TENEX SAIL

by Robert Smith

Technical Report No. 248
January 10, 1975

Institute for Mathematical Studies in the Social Sciences

Stanford University
Stanford, California
(415) 497-3797

Abstract. SAIL, a high-level ALGOL language for the PDP-10, is extended to operate under the TENEX timesharing system without executing DEC system calls. A large set of TENEX-oriented runtime routines are added to allow complete access to TENEX. The emphasis is on compatibility of programs across time-sharing systems and integrity of the language.

The research reported here was supported by the Jet Propulsion Laboratory under contract 953857 to Stanford University. This report serves both as documentation for TENEX SAIL and as the final report to the Jet Propulsion Laboratory.

Table of Contents

Acknowledgments

The author would like to express his thanks to the following people for their assistance: Jerry Feldman, Dan Swinehart, Robert Sproull, Russell Taylor, John Reiser, and Jim Low of the Stanford AI Lab; Pentti Kanerva, Alex Cannara, Bill Sanders, and Rainer Schulz of IMSSS; David Levine of Rutgers University; Nancy Smith of SUMEX; Jim Goodwin and Bill Merriam of BBN; and Meir Weinstein and John Wedel of the Jet Propulsion Laboratory. (Some of these people are at different places now.)

This document was written in PUB (thanks to Larry Tesler and Brian Harvey) and was produced on the XGP at the Stanford Artifical Intelligence Laboratory.

Support for this project was obtained from the Jet Propulsion Laboratory under Contract JPL 953857 to Stanford University.

1 *Introduction*

This reports on the modifications provided to the SAIL system (see Feldman, Swinehart and Sproull, VanLehn) to allow its use on the TENEX timesharing system. The reader is assumed to have some familiarity with the SAIL manual [3], the PDP-10 reference manual [1], and the TENEX JSYS manual [2].

An effort has been made to maintain maximum compatibility with the philosophy of the SAIL language as it runs at the Stanford Artificial Intelligence Laboratory (SU-AI), and at the same time to extend the utility and efficiency of SAIL to the TENEX timesharing system.

DEC-SAIL (as we shall refer to the SAIL implemented for the DEC system) allows the user close contact with the DEC timesharing system. TENEX-SAIL likewise has a complete set of TENEX oriented routines. Routines that are named after a JSYS (e.g., GTJFN) generally just execute that JSYS, although sometimes it is necessary to do a bit more for the sake of clarity. Several routines (e.g., OPENFILE and INDEXFILE) are a bit higher level than JSYSes, and allow the user to avoid TENEX bit-twiddling for the common operations. For data transfer routines, the emphasis has been on preserving the syntax and semantics of the old routines, for example INPUT, ARRYOUT. These routines are closely related to the data types available in the SAIL language.

Routines marked with a "‡" in the index are oriented to the DEC system and are included in TENEX SAIL for compatibility.

Routines that are named after a JSYS (and hence do not necessarily perform an exact, logical function) are marked with an "*" in the index.

For compatibility with old programs, the routines LOOKUP, ENTER, RELEASE et al. have been implemented without calling the 10-50 emulator, and are described in Section 3. We have run a large number of old DEC-style programs on TENEX-SAIL without much modification using these TENEXized routines.

Programs can use the DEC style and TENEX style routines together with few problems. For example, one might open a file with OPENFILE, do a magtape operation with MTAPE, and close the file with RELEASE.

The compiler has also been completed, and the entire set of changes has been made into the master source files at the Stanford AI Lab. Thus, users will be able to depend on the immediate availability of new SAIL features in the TENEX version. For a description of the command language changes see Section 2.

2 Operation of TENEX SAIL

2.1 The Source Code

The intent of TENEX SAIL is to make basically no changes in the structure of the language. Thus, all features are the same across systems. Source code differences are limited to:

(1) START!CODE and QUICK!CODE: the DEC uuo's (CALLI, LOOKUP etc.) are not available in TENEX SAIL. In their place are the JSYSes. Each TENEX site can easily construct its own JSYS table from the standard TENEX file <SUBSYS>STENEX.MAC (or .FAI) using a utility program MAKTAB.TNX. Details of this are (to be) in TELLEM, the implementer's guide.

(2) REQUIRE SOURCE!FILE: the name of the SOURCE!FILE may be either a TENEX file descriptor or a pseudo-DEC descriptor, i.e., any of the following:

```
<SMITH>FOOOOOOO.BAZ;3
FOOOOOOO.BAZ[SM,ITH]
FOOOOOOO.BAZ;3[SM,ITH]
```

If the source!file is not found, the user can give the name from the terminal.

(3) REQUIRE LOAD!MODULE and REQUIRE LIBRARY specifications. Note that the LOAD!MODULE and LIBRARY names must be passed to the loader, in compatibility format, and hence the name field must be limited to six characters, and the extension field to 3. (The extension is usually .REL.)

For the directory (or PPN) field, most TENEX sites have the strange BBN convention that the PPN field is of the form

```
[0,DIRNO]
```

where DIRNO is the number of the user's directory. Rather than perpetuate this error, TENEX SAIL translates the directory (or PPN field) into the number to be passed to the loader. Hence,

```
<SMITH>RELFIL
RELFIL[SM,ITH]
```

are passed to the loader with the PPN field converted to the directory number.

A few sites (e.g., IMSSS) have used sixbit PPN's with their 10/50 emulator, so that DEC cusps do a reasonable thing. For these sites, TENEX SAIL also does the right thing, under conditional compilation of the sources.

(4) The additional runtime routines described in this document are defined to the compiler in TENEX SAIL. For those users desiring to write programs that conditionally compile according to whether or not the site is a TENEX installation, some compile-time test such as

```
IFC NOT DECLARATION(GTJFN) THENC ....
```

will work, since GTJFN is not defined in DEC SAIL.

2.2 *Compiler Command Language*

The command language to the compiler is the most different thing about TENEX SAIL. It has been redesigned to look like a BBN style TENEX program.

When you say

```
@SAIL
```

it prints back

```
   TENEX SAIL 8.1  1-10-75  (? for help)

*
```

The "8.1" is the version number and it is followed by the compiler creation date.

The asterisk is the top command level. Two asterisks are for subcommand level.

At any point that TENEX SAIL requests a command or subcommand, the following are legal:

```
?         options available at this level
control-Q
          quit
control-U
          restart
```

A command is basically a list of files to be compiled as a concatenated source stream. File name recognition is available, and the default extension is .SAI.

Devices available are TTY:, DSK: and DTAn:. Other devices do not currently work in the compiler (although they work in the runtime routines described later.) Device names are NOT "sticky", as they are in DEC SAIL, and the default device is always DSK:. If the device is TTY:, then a bare control-Z terminates input. Editing is available with control-X (delete the current line) control-R (retype the current line) and either control-A or rubout (delete a character).

The assumption is made that the user wants to compile his file(s) into a .REL file with the same name as the first file. (Note that the standard DEC cusp specification

```
FOO←FOO
```

is usually redundant.) If the user begins his command with a left arrow, no binary file is made. The name of the .REL file can be changed by a subcommand. Also, the listing file and the switches are subcommands.

The following is a BNF definition of the command language, with the semantics following the semicolon. <FILE> is a TENEX file specification (done by GTJFN with recognition), and <cr> is a carriage return.

```
<COMMAND>::=  <FILELIST><cr>     ;compile files in <FILELIST>
<COMMAND>::=  <FILELIST>,<cr>    ;compile, subcommand mode
<COMMAND>::=  <FILELIST>←        ;compile, load with DDT
                                 ;if available
<COMMAND>::=  <FILELIST>,←       ;compile, subcommand, load
                                 ;with DDT if available
<COMMAND>::= ←<FILELIST>         ;compile, no .REL file

<FILELIST>::=  <FILE>
<FILELIST>::=  <FILE>,<FILELIST>
```

After giving the command, if subcommand level was requested, the compiler prints

**

to indicate subcommand level. The subcommand syntax is

```
<SUB>::= <cr>                    ;bare carriage return to
                                 ;start compilation
<SUB>::= <control-R>             ;control-R -- .REL file
                                 ;specification via GTJFN
<SUB>::= <control-L>             ;.LST specification to GTJFN

<SUB>::= / <SWITCH>              ;some switch

Valid switches are:

G       load after compilation, exiting to the EXEC
T       load with DDT
R       double parse stack
C       cref listing (listing filename must be specified first!!)
D       double define pdl
P       double system pdl
Q       double string pdl
H       sharable compilation (default on TENEX)
I       non-sharable compilation
K       Kount feature
<NUM>B  BAIL features
<NUM>S  string space
<NUM>F  listing format

            [See Sec. 19.3  of [3] for a more detailed
                 description of these switches.]
```

Additional debugging features for the compiler hacker are available as subcommands if the compiler is a debugging version.

2.3 *Loading SAIL Programs*

The loader interfaces are available at IMSSS, and available at other TENEX sites provided the TMPCOR feature exists in the emulator.

To load a SAIL program, the main trick is to load the "SAILOW" file first. In TENEX, the name of this file (to the loader) is "SYS:LOWTSA.REL", where "SYS:" is whatever directory the emulator recognizes as such (this is now usually <SUBSYS>). Thus, the following are some reasonable command strings to LOADER (where "$" is an altmode):

```
SYS:LOWTSA,DSK:FOO$                ;load program FOO
SYS:LOWTSA,DSK:FOO,BAZ$            ;load programs FOO and BAZ
SYS:LOWTSA,/TDSK:FOO$              ;load with DDT
```

At sites where LINK10 is available, (such as SUMEX and PARC) it is strongly recommended to load with LINK10. A sample command sequence would be:

```
@LINK10
*SYS:LOWTSA                        ;load crucial file first!!
*DSK:FOO                           ;program foo
*/G                                ;exit to the EXEC
```

LINK10 is considerably faster than LOADER; further, BBN has apparently straightened out the problems of the state of the core image at the end of the loading process in LINK10 running on version 1.31 of TENEX. In LOADER, there are several hassles pertaining to the state of the PSI system and the emulator. Warning: LINK10 may not work with some versions of DEC FORTRAN if the SAIL program calls FORTRAN subroutines.

2.4 *Editor Interfaces*

Interfaces to all editors are in the IMSSS version only. "T" gets the "best" editor for whatever terminal you are on, "E" gets STOPGAP. At IMSSS, the editor interfaces are currently limited to 6-character names and 3-character extensions, with no version specification.

2.5 *Compiler Error Handling*

The error handler works as in DEC SAIL. Logging features are available, with the default name FILE.LOG, where FILE is the (first) source name. In specifying the log file name from the terminal, GTJFN is used with recognition.

The Stanford escape-I interrupt (to reset the error handler) is implemented with a TENEX pseudo-interrupt on the character control-H. (Control-I would have been a bad choice since that character is a TAB.) Typing a control-H resets the error handler when it has been put into auto-continue mode.

3 *DEC-Style Runtime Routines*

The design criterion of the new routines is to allow "standard" programs written for the DEC I/O to run on TENEX-SAIL without the emulator. The routines described in this section are the old routines. Refer to the SAIL manual [3] for a complete description of these routines as they originally worked.

Some routines (e.g., LOOKUP) return error numbers. In TENEX-SAIL, these error numbers are the TENEX error numbers (in the sense of the ERSTR JSYS) rather than the DEC error numbers. The justification for this is that (1) few SAIL programs check, for example, the nature of a LOOKUP error (illegal PPN, illegal file, file being modified, etc); and (2) any program that was examining information that closely would require recoding using the TENEX error numbers, which apply more sensibly to a job running on TENEX.

The CALL routine described on page 8 is a somewhat special function, in that in DEC-SAIL it is intended to provide special system calls. We have implemented in TENEX-SAIL those CALLs that are most likely to be used in a user SAIL program dating from the period of time that SAIL has been available at IMSSS.

OPEN(*CHAN,"DEV",MODE,IBUF,OBUF,@COUNT,@BRK,@EOF*)

In the DEC system, the *MODE* parameter indicates which of the data modes are to be used. Typical choices are: 0 for characters, '14 for 36-bit words, '17 for dump mode. In TENEX, this parameter is mostly ignored. In TENEX it is not, for example, possible to do dump mode I/O to the dsk, nor is it possible to do character mode I/O to the magtape. A case that is not ignored: dump mode I/O to the dectape gives dump mode as on the DEC system, wherein one can ignore the directory on the tape.

In the DEC system, the *IBUF* and *OBUF* parameters specify the number of buffers desired for input and output respectively. Since buffering is handled differently in TENEX, these are used only to indicate whether input or output is desired.

Currently, the above two facts create an incompatibility, which occurs when a device is opened in dump mode with no buffers specified and then a data transfer attempted without either a LOOKUP or ENTER. This bug will be corrected; until it is, the user should see the OPENFILE routine, which is intended to replace OPEN, LOOKUP, and ENTER for file accessing in TENEX SAIL.

LOOKUP(*CHAN,"FILE",@FLAG*)

For the "*FILE*" parameter, LOOKUP, ENTER, AND RENAME in TENEX-SAIL all accept either DEC or TENEX file specification, as does REQUIRE SOURCE!FILE (see Section 2.1 above).

ENTER(*CHAN,"FILE",@FLAG*)

Simulates the ENTER function.

RENAME(*CHAN,"NEW!NAME",PROTECTION,@FLAG*)

The *PROTECTION* feature does not work. Note that TENEX provides extensive protection features which are accessible through the GTFDB and CHFDB routines.

USETI(*CHAN,BLOCK*)

On TENEX, there is only one file pointer for both input and output; hence, USETI and USETO are the same function in TENEX SAIL. USETI (or USETO) only works on those devices where the SFPTR JSYS works. Currently, this is only the disk in TENEX. A USETI to a dectape causes the block on the tape to be set via the MTOPR JSYS, which is not necessarily what is intended on the DEC system.

In DEC SAIL, the action of USETI could be somewhat confusing, since the effect of the USETI only takes place when the next input from the system is requested. Thus, old data still in buffers can be input before the effect of the USETI is felt. TENEX SAIL has random input and output more completely implemented. Hence, USETI and USETO take place immediately, with the next input or output referencing *BLOCK*.

See the discussion of random I/O in section Subsection 5.3, noting that `USETI(CHAN,BLOCK)` is equivalent to `SWDPTR(CHAN,(BLOCK-1)*128)`.

USETO(*CHAN,BLOCK*)

TENEX has only one file pointer (whereas the DEC system in most of its incarnations has two). Thus, USETO is the same function in TENEX SAIL as USETI.

CLOSIN(*CHAN*)

Simulates the CLOSIN function.

CLOSO(*CHAN*)

Simulates the CLOSO function.

CLOSE(*CHAN*)

Does both a CLOSIN and a CLOSO.

RELEASE(*CHAN*)

Simulates the RELEASE function.

MTAPE(*CHAN,FUNCTION*)

FUNCTION is performed on *CHAN*, according to the following code:

FUNCTION	Operation performed
"A"	Advance past one tape mark (or file)
"B"	Backspace past one tape mark
"E"	Write tape mark
"F"	Advance one record
"R"	Backspace one record
"S"	Write 3 inches of blank tape
"T"	Advance to logical end of tape
"U"	Rewind and unload
"W"	Rewind tape to load point

In the current DEC version of SAIL, another mode has been implemented: "I" for IBM-compatible mode. Since this is not a standard TENEX feature, it is not available.

The user should also see the MTOPR routine, which does the TENEX MTOPR JSYS in its full generality. In particular, some DECTAPE operations are performed with MTOPR.

The user is warned that there are serious limitations in TENEX regarding magtapes. While TENEX is supposed to have device-independent I/O, the magtape code in TENEX (as of v. 1-31) is minimal, allowing only dump mode transfers. Further, end of file markers must be written explicitly, and it is sometimes necessary to do an MTOPR operation 0 to reset the magtape status bits.

TENEX SAIL has been designed to handle some of these things in a way that makes features available on a standard DEC system available in a transparent way. For example, string input and output functions work, with SAIL assuming 128-word records on the tape. ARRYIN and ARRYOUT cause the DUMPI and DUMPO JSYSes to be executed for the specified word counts. Closing a magtape file open for writing will cause two EOF's to be written, and one record backspaced. Status bits are cleared automatically. Thus, sequential reads and writes will work pretty much as they did on a DEC system. The user who wants to write magtape code for operations other than the above is hereby warned that the TENEX magtape code is frought with peril. TENEX SAIL certainly allows full access to TENEX in this regard, however.

NEW!CHAN ← GETCHAN

GETCHAN in TENEX corrects a bug still present in DEC-SAIL, namely, that while GETCHAN returns a channel thought to be available, it does not reserve that channel, and hence successive GETCHANs without intervening OPENs will get the same channel. In TENEX-SAIL, GETCHAN reserves the channel until an OPEN or RELEASE is performed.

REAL!JFN ← CVJFN(*CHAN*)

This is an important function conceptually, although you may never need it. SAIL uses channels for both the DEC style I/O and the TENEX style I/O. For most purposes, you do not need to know what the TENEX JFN "really" is, since all the routines in TENEX-SAIL use the channel numbers. But, if you really want to have the 36-bit JFN (including flags in the left half), this function will provide that information.

RESULT ← CALL(*ACCUMULATOR!ARGUMENT*,"*FUNCTION*")

The CALL function in SAIL was designed to call whichever CALLIs were defined in the particular time-sharing system. The following CALLIs are supported (in pure TENEX coding) in the TENEX-SAIL. Any other call will give a non-fatal SAIL error, from which continuation is possible.

Implemented CALLIs:

```
EXIT
DATE
LOGOUT
TIMER
MSTIME
GETPPN
RUNTIM
PJOB
RUN
```

In addition, the IMSSS version has the following special CALLIs defined.

```
DATSAV
PUTINF
GETINF
RANDOM
```

It is believed that this includes most of the CALLIs likely to be made from a SAIL program.

4 *Obtaining and Releasing Files*

The routines in this section replace the DEC routines OPEN, LOOKUP, ENTER and RELEASE. They allow these operations to be done, relative to the SAIL data structures, with much of the full generality of TENEX.

The following functions are the standard tools to reach for in doing I/O. The new user to TENEX SAIL will find that these satisfy most SAIL and TENEX needs.

OPENFILE	Obtain a file
CFILE	Release a file
SETINPUT	Set parameters for input
INPUT	Read in a string
OUT	Write a string
ARRYIN	Read in an array (36-bit words)
ARRYOUT	Write an array
JFNS	Read file name

4.1 Main File Opening Routines

The main procedure is OPENFILE. In terms of TENEX, OPENFILE does a GTJFN and OPENF. Additional routines provide support to OPENFILE including SETINPUT, SETPL, INDEXFILE, and CFILE.

CHAN ← OPENFILE("*NAME*", "*OPTIONS*")

NAME is the name of the file to be opened. If it is null, then OPENFILE goes to the user's console for the filename, using TENEX filename recognition.

CHAN, the value of the call, is a SAIL channel number. Note that the channel number is not necessarily the same as the TENEX JFN (although it probably is). If you really need to know the JFN, use CVJFN. All TENEX SAIL functions accept a channel number, not a JFN. (Exception: SETCHAN, for hackers only.)

OPTIONS is a string of options available to the user. Legal options are:

```
Read or write:
         R                read
         W                write
         A                append
Version numbering, old-new:
         O                old file
         N                new file
         T                temporary file
         *                index with INDEXFILE routine
Independent bits to be set:
         C                require confirmation
         D                ignore deleted bit
         H                "thawed" access
Error handling:
         E                return errors to user in the external
                          integer !skip!.  TENEX error codes are used.
                          (CHAN will be released in this case.)
```

OPENFILE does a GTJFN followed by a OPENF. If some error occurs with the file, an error message is printed by OPENFILE, and an attempt is made to obtain a file name, from the user's console. This internal handling of errors can be avoided with the "E" mode. If an error occurs when the "E" mode was set, OPENFILE will return -1 to the user, and the TENEX error code will be put into EXTERNAL INTEGER *!skip!*. Examples of the OPENFILE routine follow. To write on a file, the name of which is specified from the terminal:

```
BEGIN
INTEGER JFN;
OUTSTR("FILE NAME* ");
JFN <- OPENFILE(NULL, "WC");  COMMENT write, confirm name;
OUT(JFN, "text
");
CFILE(JFN);                   COMMENT close the file;
END;
```

To read lines from a file:

```
BEGIN
STRING S;
INTEGER JFN, BRCHAR, EOF;
SETBREAK(1,'12,'15&'14,"IN");
OUTSTR("INPUT FILE* ");
JFN <- OPENFILE(NULL,"RCO");
SETINPUT(JFN,200,BRCHAR,EOF);
DO
    BEGIN
        S <- INPUT(JFN,1);
    END UNTIL EOF;
CFILE(JFN);
END;
```

SETINPUT(*CHAN*, @*COUNT*, @*BRCHAR*, @*EOF*)

This routine relates to the functions of the last three arguments of the OPEN function in DEC-SAIL. When the user wishes to use the *COUNT*, *BRCHAR*, and *EOF* features of the input routines, SETINPUT will associate the reference locations *COUNT*, *BRCHAR*, and EOF with the input from *CHAN*. The INPUT function (see Section 5.2) uses 200 as the default value of *COUNT* if no location has been associated with *CHAN*.

As an added convenience, all I/O transfer routines set *!skip!* to indicate end-of-file and I/O errors. For example, on return from INPUT (see Section 5.2), *!skip!* will be -1 if an end-of-file occurred, a TENEX error number if an error occurred, or 0 otherwise.

SETPL(*CHAN*, @*LINNUM*, @*PAGNUM*, @*SOSNUM*)

This routine extends the usefulness of the INPUT function in reading through a file. Often, one wants to read a file, but be able to keep track of where one is in the file. This routine allows the user to name the variables to be used by the INPUT function for counting the linefeeds ('12) and formfeeds ('14) seen by INPUT, as well as keeping the current SOS line number, if any. SETPL initializes all three variables to 0. Each time a formfeed is seen, *LINNUM* is set to 0, and *PAGNUM* is incremented. Hence, *LINNUM* is a count of the number of linefeeds seen on the current page, and *PAGNUM* is the number of formfeeds seen.

Only input with the INPUT, REALIN, or INTIN functions is affected. In particular, CHARIN, and SINI do not affect these locations. CHARIN and SINI were coded for speed rather than flexibility.

FOUND!ANOTHER!FILE ← INDEXFILE(*CHAN*)

This routine is useful when OPENFILE is used with the "*" option, which indexes through many files. (This is the way that the "DIRECTORY" command works in TENEX.) INDEXFILE returns TRUE as long as another file can be found on *CHAN*.

The following SAIL statements illustrate the use of OPENFILE, SETINPUT, and INDEXFILE.

```
JFN ← OPENFILE("<JONES>*.SAI;*","RO*");
        COMMENT Read all of Jones's SAIL programs.;
SETINPUT(JFN,200,0,EOF);

DO
   BEGIN "INDEX"

      DO
       BEGIN "READ FILE"
        STRING S;
        S ← INPUT(JFN,BREAK!TABLE);
        COMMENT process ...;
       END "READ FILE" UNTIL EOF;

   END "INDEX" UNTIL NOT INDEXFILE(JFN);
```

For those familiar with the DEC system, the "*" option takes the place of reading the MFD and the UFD's. INDEXFILE clears the *EOF*, *LINNUM*, *SOSNUM*, and *PAGNUM* variables if these have been set by SETINPUT and SETPL.

4.2 *Auxiliary File Opening Routines*

CHAN ← GTJFN("*FILENAME*",*FLAGS*)

Does a GTJFN. If *FILENAME* is non-null, it is the filename, otherwise the routine goes to the user's console for a file. *FLAGS* are used for accumulator 1, and any error code is returned in *!skip!*. The value of the call is the *CHAN*, if obtained.

Ordinarily, this routine will not be needed, since the more facile routine OPENFILE is available. (GTJFN, OPENF, GNJFN, CLOSF, RLJFN, and DVCHR are all in this category of being included only for completeness, and are not necessary for most I/O needs.)

MORE!FILES ← GNJFN(*CHAN*)

Does the GNJFN JSYS. Note that a file that is open cannot have GNJFN applied to it. The user will usually find that the INDEXFILE, which executes the GNJFN JSYS among others, is actually the logical function he desires. The exception is if files are being indexed without actually being opened (i.e., without an OPENF JSYS), which is a sensible way of performing some operations, such as counting the number of files in a group.

CHAN ← GTJFNL
("*ORIGSTR*",*FLAGS*,*JFN!JFN*,"*DEV*","*DIR*","*NAM*",
"*EXT*","*PROT*","*ACCOUNT*",*DESIRED!JFN*)

Does the long form of the GTJFN JSYS. The arguments are put into the accumulators and locations in the table accepted by the long form. (See [2] for a description of the long form of GTJFN.)

These arguments are given below, where "AC X" means an accumulator, and "E+X" means in the Xth address of the table.

Argument	Where placed	What
"ORIGSTR"	AC 2	Partial or complete string
FLAGS	E+0	Flags to GTJFN
JFN!JFN	E+1	xwd input JFN, output JFN
The following strings may be defaulted to NULL		
"DEV"	E+2	device
"DIR"	E+3	directory
"NAME"	E+4	name
"EXT"	E+5	extension
"PROT"	E+6	protection
"ACCOUNT"	E+7	account
DESIRED!JFN	E+'10	desired JFN if B11 on

It should be noted that GTJFNL will not OPENF the file. See the OPENF routine.

OPENF(*CHAN,FLAGS*)

Does the OPENF JSYS on *CHAN*, with *FLAGS* the contents of accumulator 2. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

The occasion may arise that the user will find the options in the OPENFILE routine to be insufficient, and will need to use either GTJFN or GTJFNL to get the JFN. In these cases, it will usually be necessary to OPENF the file with this routine.

The user should note that the ways in which TENEX-SAIL opens files do not always correspond to what would be obvious. This is in part due to the need to have all 36 bits whenever the device is capable of transmitting 36 bits, so that STOPGAP line numbers work in the INPUT and LINOUT functions. Best results are obtained by opening a TTY in 7-bit mode, the DSK or DTA in 36-bit mode, and a magtape in 36-bit dump mode. (Indeed other possibilities may not even work: for example, magtapes cannot currently be opened in mode 0 in TENEX.)

CHAN ← SETCHAN(*REAL!JFN, GTJFN!FLAGS, OPENF!FLAGS*)

This function should be considered liberation from SAIL I/O. It is provided for those who wish to do SAIL I/O from a JFN that is obtained from some other means than the SAIL file-opening routines -- for example, a JFN passed from a superior fork.

REAL!JFN is a 36-bit JFN (or JFN substitute, such as a Teletype number), *GTJFN!FLAGS* and *OPENF!FLAGS* are the flags that should be recorded describing how the GTJFN and OPENF were accomplished. REAL!JFN need not be open. The value returned by

SETCHAN is a TENEX SAIL channel number, which should be used for subsequent TENEX SAIL I/O. SETCHAN is the only function in TENEX SAIL that takes an actual JFN as an argument.

4.3 *File Closing Routines*

SUCCESS ← CFILE(*CHAN*)

Closes the file (CLOSF) and releases (RLFJN) the *CHAN*. This is the ordinary way the user will dispense with a file.

Returns TRUE if *CHAN* legal and released and returns FALSE otherwise.

CLOSF(*CHAN*)

Does a CLOSF on *CHAN*. Ordinarily the user will want to use the CFILE routine, which handles errors internally. The CLOSF is accomplished in such a way that *CHAN* is actually not released.

If the device is a magtape open for output, then 2 end-of-file marks are written, followed by a backspace. This writes a standard end-of-file on the tape.

CLOSF therefore does more than its name implies.

RLJFN(*CHAN*)

Does the RLJFN JSYS. Ordinarily the user will want to use the CFILE routine.

4.4 *Auxiliary File and Device Routines*

DELF(*CHAN*)

Deletes file on *CHAN*. The file must not be open at the time, so you may have to do a CLOSF first. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

NUMBER!DELETED ← DELNF(*CHAN*, *NUMBER!RETAIN*)

Deletes all but *NUMBER!RETAIN* versions of the file on *CHAN*, which must be CLOSFed first. Thus, letting *NUMBER!RETAIN* be 0 will delete all versions of the file, *NUMBER,RETAIN* 1 will keep only the most recent version of the file. The number of files actually deleted is returned as the value of the call.

UNDELETE(*CHAN*)

Undeletes the file open on *CHAN*. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

SIZE ← SIZEF(*CHAN*)

Gets the size in pages of the file open on *CHAN*. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

"*FILENAME*" ← JFNS(*CHAN*,*FLAGS*)

Returns the name of the file associated with *CHAN*. *FLAGS* are for accumulator 3 as described in the JSYS manual. 0 is the reasonable default for *FLAGS*.

STATUS ← GTSTS(*CHAN*)

Gets the file status with the GTSTS JSYS.

WARNING: The results of this call are not necessarily appropriate for determining the end-of-file if the file is being page-mapped by SAIL. If you want to check for end-of-file, examine the EOF variable instead. See SETINPUT.

STSTS(*CHAN*,*NEW!STATUS*)

Sets the status of file on *CHAN* to *NEW!STATUS*. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

SUCCESS ← RNAMF(*EXISTINGCHAN*,*NEWCHAN*)

Does the RNAMF JSYS, renaming the file on *EXISTINGCHAN* to the name of the (vestigal) file on *NEWCHAN*. It is necessary for the user to CLOSF *EXISTINGCHAN* before this operation, and then OPENF again afterwards.

The routine RENAME from the DEC-style routines is sometimes more convenient to use than RNAMF, since it performs the GTJFN and OPENFs necessary for the renaming operation. However, the actual JFN associated with CHAN is of course changed by RENAME.

SUCCESS ← ASND(*DEVICE*)

DEVICE (a TENEX device descriptor) is assigned to the user. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

SUCCESS ← RELD(*DEVICE!DESCRIPTOR*)

DEVICE is deassigned. If *DEVICE* is -1, then all devices assigned to the job are deassigned. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

DEV!STAT ← GDSTS(*CHAN*,*@WORD!COUNT*)

The status of the device on *CHAN* is returned in *DEV!STAT* and the contents of AC 3 (the word count) is returned in reference variable *WORD!COUNT*.

SDSTS(*CHAN*,*NEW!STATUS*)

The status of the device on *CHAN* is changed to *NEW!STATUS*. Remark: some magtape statuses (such as EOF) are not, as one would suspect, set by this JSYS, but rather by the MTOPR JSYS. Ordinarily, the SAIL runtime system takes care of this, but it is worth mentioning, since so many users have run into this poorly documented fact about TENEX.

DEVICE!DESIGNATOR ← STDEV("*DEVICE!NAME*")

A string (such as "DTA0") is converted to a *DEVICE!DESIGNATOR*. Warning: TENEX does not accept lower case characters as being equivalent to upper case characters on the STDEV JSYS (although it does on GTJFN for example.) TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

"*DEVICE!NAME*" ← DEVST(*DEVICE!DESIGNATOR*)

Returns the string name of *DEVICE!DESIGNATOR*.

DEVICE!TYPE ← DEVTYPE(*CHAN*)

Returns (via the DVCHR JSYS) the device type of the device open on *CHAN*. The more general DVCHR call is also implemented (below).

DEVICE!CHARACTERISTICS ← DVCHR(*CHAN*,@*AC1*,@*AC3*)

Does the DEVCHR JSYS, returning the flags from AC2 as the value of the call, and *AC1* and *AC3* get the contents of ac's 1 and 3.

GTFDB(*CHAN; REFERENCE INTEGER ARRAY BUF*)

Entire FDB of *CHAN* is read into *BUF*. No bounds checking, so *BUF* should be at least '25 words.

CHFDB(*CHAN, DISPLACEMENT, MASK, CHANGED!BITS*)

Does the CHFDB JSYS on *CHAN*, with *DISPLACEMENT*, *MASK*, and *CHANGED!BITS* as described in [2].

MTOPR(*CHAN,FUNCTION,VALUE*)

Does the MTOPR JSYS. The user may find the DEC-style MTAPE function more comfortable. *FUNCTION* goes into accumulator 2, and *VALUE* goes into accumulator 3

BKJFN(*CHAN*)

Does the BKJFN JSYS on *CHAN*. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred. Not intended for ordinary use.

BYTE-SIZE ← RFBSZ(*CHAN*)

Reads the byte-size of the file open on *CHAN*. Not intended for normal use.

5 *Data Transfer Routines*

The routines described in this section do data transfers. For the most part, these routines are the same syntactically and semantically as the routines in DEC-SAIL.

One improvement in TENEX SAIL is that characters and words can be mixed in reading or writing to a file, provided the file is on the disk. Such I/O is called *data mixed I/O*.

The following interpretation is given to data mixed I/O. There is one logical character pointer into the file. When a character is read or written, it accesses the byte pointed to by the pointer. There is only one pointer for both input and output. When a word is read or written, the next full word position in the file is accessed.

See below for reading and changing this pointer. This feature is only available to those devices that support random I/O, currently only the disk.

5.1 *Output Routines*

CHAROUT(*CHAN*, *CHAR*)

Writes a single *CHAR* to *CHAN*.

OUT(*CHAN*,"*STRING*")

Outputs a SAIL string to the *CHAN*.

LINOUT(*CHAN*, *VALUE*)

This routine has little use, but is included since it was in the DEC-SAIL. It writes an SOS line number on *CHAN*.

WORD ← WORDIN(*CHAN*)

Reads a word (36-bits) from the file, which must be open for 36-bit transfers.

WORDOUT(*CHAN*, *BYTE*)

Sends a 36-bit *BYTE* to *CHAN*, which must be open for 36-bit transfers.

ARRYIN(*CHAN*, @*LOC*, *COUNT*)

Reads in *COUNT* words into *LOC* from *CHAN*. The file should be open for 36-bit bytes for this to work. If the *CHAN* is open in TENEX DUMP mode (which should not be confused with the DEC system DUMP mode), each ARRYIN or ARRYOUT specifies a single DUMPI/DUMPO operation. This is particularly relevant to magtape operations.

WARNING: no array bounds checking.

ARRYOUT(*CHAN*, @*LOC*, *COUNT*)

Writes *COUNT* words to *CHAN* starting at *LOC*. The file should be open in 36-bit bytes.

5.2 *Character Input From Files*

DEC-SAIL had the following routines for character input:

```
INPUT REALIN INTIN
```

TENEX-SAIL has these routines, plus the routines CHARIN (read a single character) and SINI (read a string upto a specified character). Also, the INPUT function can be set (using SETPL) to count the page and line numbers in the file.

CHAR ← CHARIN(*CHAN*)

Returns the next character from a file (open for character reading). Returns 0 if the file is at the end.

"*INPUT!STRING*" ← SINI(*CHAN*, *MAXLENGTH*, *BRKCHAR*)

Reads in a string of characters, terminated by *BRKCHAR* or reaching *MAXLENGTH*, whichever happens first. (It is named SINI because it is like the SIN JSYS in TENEX.)

The external integer *!skip!* will be -1 if call terminated for count, else it will have the breakcharacter, which will be the last character read if the end-of-file is encountered.

"*INPUT!STRING*" ← INPUT(*CHAN*, *BREAK!TABLE!NUMBER*)

REAL ← REALIN(*CHAN*)

INTEGER ← INTIN(*CHAN*)

[For a description of INPUT, REALIN, AND INTIN see Sec 7.4 of [3].]

STDBRK(*CHAN*)

This routine reads in the standard break-table file on *CHAN*.

5.3 *Random Input and Output*

As mentioned previously, TENEX SAIL allows the mixture of input and output from the same file with both characters and words being read and written. For these purposes there is one logical character pointer to the next character to be read or written. Either reading or writing a character increments that pointer. If a read passes the end of the file, the EOF variable in the SETINPUT function (or the DEC function OPEN) and external integer *!skip!* are set to -1. If a write passes the end of file, then the end of file is advanced. Reading or writing a word advances the character pointer to the next full word in the file, where 5 ASCII characters is of course one word.

When the device allows for random I/O, TENEX SAIL provides the following four logical functions to read and modify the character and word pointers into the file:

```
RCHPTR(CHAN)                    reads the character pointer
SCHPTR(CHAN,POINTER)            sets the character pointer
RWDPTR(CHAN)                    reads the word pointer
SWDPTR(CHAN,POINTER)            sets the word pointer
```

RCHPTR and SCHPTR read and change the character pointer. RWDPTR and SWDPTR read and change the word pointer. These functions work like RFPTR and SFPTR when the file is open in 7-bit size for characters or 36-bit size for words. The user should not, however, use RFPTR or SFPTR directly, since the regular TENEX pointer accessed by RFPTR and SFPTR will not necessarily reflect buffering.

The TENEX SAIL user is warned that data-mixed I/O and random I/O are not features of DEC SAIL; hence, programs using these features will not be readily transportable to PDP-10 sites using the DEC system.

Currently the random feature only works on the disk; however, they are coded so that if RFPTR and SFPTR are ever extended to other devices (e.g., DEC tape), then random I/O would work on those devices as well. For fastest operation on the disk, the appropriate pages of the disk file are mapped using the PMAP JSYS.

There is some inefficiency associated with data mixed I/O and random I/O, since the SAIL runtime system has to shuffle its pointers and perhaps do PMAPs and reset the end-of-file pointer. Calls to TENEX (which represent the greatest potential overhead) are however kept to a minimum for fastest operation.

The following is the list of routines that are associated with random I/O. The routines RFPTR and SFPTR are included for completeness, and should not be used by the ordinary user. The character pointer is accessed with RCHPTR and SCHPTR, and the word pointer is accessed with RWDPTR and SWDPTR.

CHARPOINTER ← RCHPTR(*CHAN*)

Returns the byte that will next be accessed by a character read or write operation, where the first character is character number 0. Illegal unless the file is on the disk. Does not change the state of the I/O system. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

SCHPTR(*CHAN,NEWPOINTER*)

Changes the byte that will next be accessed by a character read or write operation. If

NEWPOINTER is -1, then the pointer is set to the end of file. Setting the pointer beyond the end of the file may change the length of the file if it is being written. Illegal unless the file is on the disk. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

WORDPOINTER ← RWDPTR(*CHAN*)

Returns the number of the next word that will be read or written by a word operation, where the first word is word number 0. Illegal unless the file is on the disk. Does not change the state of the I/O system. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

SWDPTR(*CHAN,NEWPOINTER*)

Changes the word that will next be accessed by a word read or write operation. If *NEWPOINTER* is -1, then the pointer is set to the end of the file. Setting the pointer beyond the end of the file will change the length of the file if it is being written. Illegal unless the file is on the disk. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

SFPTR(*CHAN,POINTER*)

Does the SFPTR JSYS. Not intended for normal use. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

POINTER ← RFPTR(*CHAN*)

Does the RFPTR JSYS. Not intended for normal use. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

5.4 Direct DSK Operations

The routines DSKIN and DSKOUT do direct DSK operations in TENEX-SAIL. They correspond to the device PAK in the DEC system. These routines relate only to the IMSSS version of TENEX-SAIL.

DSKIN(*MODULE, RECNO, COUNT, @LOC*)

[IMSSS only.]

DSKIN does direct I/O from the DSK (formerly device "PAK"). Modules 4-7 are legal for everyone, other modules require enabled status.

COUNT words (=<'1000) are read into user's core at location *LOC*, from *MODULE*, record *RECNO*. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred. WARNING: No bounds checking is performed to see if the *LOC* is a legal SAIL array.

DSKOUT(*MODULE, RECNO, COUNT, @LOC*)

[IMSSS only.]

Works like DSKIN, except that it writes instead of reads. No bounds checking is performed on the locations being written (so they may not be legal SAIL arrays).

6 *Error Handling*

When errors occur, sometimes the runtime routines trap these errors themselves. This practice is held to a minimum, since the error itself may be information that the user is interested in seeing. Usually, the routines (as marked) put the TENEX error code in EXTERNAL INTEGER *!skip!*, which you may declare and examine after any call so marked. The TENEX error numbers don't always make good sense, but for the cases that they do, the ERSTR routine will print out on the user's console the message associated with that error number.

ERSTR(*ERRNO,FORK*)

Using the ERSTR JSYS, types out on the console the TENEX error string associated with *ERRNO* for fork *FORK* ('400000 for the current fork). Parameters (in the sense of the ERSTR JSYS) are expanded.

Types out the string ERSTR: UNDEFINED ERROR NUMBER if something is wrong with your error number or fork (and sets *!skip!* to -1).

7 *Terminal Handling*

The routines in this section really refer to terminals only in the so-called "mini-system" version of TENEX (planned obsolescence). The argument *CHAN* may be either a SAIL channel number associated with a terminal, or a terminal specifier (such as '100 or '101 for the controlling terminal).

MODE!WORD ← RFMOD(*CHAN*)

Reads a file's mode word, using the RFMOD JSYS.

SFMOD(*CHAN,AC2*)

Sets a file's mode word to argument *AC2*, using the SFMOD JSYS. WARNING: some features, such as upper case conversion, that are advertised by BBN as being accomplished with the SFMOD JSYS are actually accomplished with the STPAR JSYS.

RFCOC(*CHAN,@AC2,@AC3*)

Does RFCOC JSYS, returning values in *iAC2* and *AC3*.

SFCOC(*CHAN,AC2,AC3*)

Does SFCOC JSYS, setting to *AC2* and *AC3*.

STPAR(*CHAN*,*AC2*)

Does the STPAR JSYS, setting to *AC2*.

TERMINAL!TYPE ← GTTYP(*CHAN*,@*BUFFERS*)

Returns the terminal type associated with *CHAN*, additional values returned from accumulator 2 into reference parameter *BUFFERS*.

STTYP(*CHAN*,*AC2*)

Simulates terminal input on *CHAN* by putting *AC2* in the input buffer.

The usual SAIL routines for teletype I/O are all available in TENEX SAIL. These routines are:

```
        INCHRW INCHRS INCHWL INCHSL INSTR INSTRL INSTRS OUTCHR OUTSTR CLRBUF
    TTYIN TTYIN TTYINL
```

In addition, PBIN, PBOUT, and PSOUT have been added, although they execute exactly the same code as INCHRW, OUTCHR, and OUTSTR respectively. TENEX-SAIL sets the modes for teletype I/O in a way similar to the DEC system. You may examine and change those modes with the RFMOD, SFMOD, etc., described above.

CHARACTER ← PBTIN(*SECONDS*)

[IMSSS only.]

Executes the PBTIN JSYS, with timing of SECONDS. IMSSS only.

"*INPUT!STRING*" ← INTTY

INTTY does a TENEX-style input. (Note that INCHWL does a DEC system style line input, with rubout deleting one character and control-U deleting the whole line.)

At sites other than IMSSS, this routine gets a line of up to 200 characters using editing conventions similar to many TENEX programs. Control-X deletes the line, control-R repeats the current partial line, and both control-A and rubout delete a single character. Line activation is with an EOL, ESCAPE, control-Z or control-G. The break character is not appended to the string returned, but is put into *!skip!*, which is -1 if the input is terminated for overflowing the 200 character limit.

At IMSSS, this routine uses the PSTIN JSYS, and accepts as many as 200 characters from the user's Teletype, with the standard system breakcharacters. The breakcharacter itself is removed from the string, and no timing is available. If the default of 200 characters is exceeded, then *!skip!* is set to -1; otherwise, *!skip!* has the break-character.

8 *Paging the TENEX-SAIL Core Image*

Presently, the only page-manipulation routine available is PMAP. It is clearly desirable that the SAIL allocation routines (CORGET et al.) be redesigned to take advantage of the paging environment, and that certain data types be added to the SAIL language to control allocation. The only feature added is the PRESET!WITH construction, which is like PRELOAD!WITH except that the array is placed in the high segment.

PMAP(*AC1,AC2,AC3*)

Does the PMAP JSYS, using the accumulators for the arguments.

9 *Utility TENEX System Calls*

The functions in this section do utility calls on TENEX. An effort has been made to provide calls that read and write strings which may be inconvenient for the user to perform in START!CODE. It should be noted that the TENEX-SAIL compiler has the TENEX JSYS mnemonics defined in START!CODE. These definitions should not be confused with the function calls of the same name.

"*DATE!TIME*" ← ODTIM(*DT,FORMAT*)

The string representation of the date and time *DT* are returned in format specified by *FORMAT*. *DT* is in internal TENEX representation.

```
TENEX Defaults:

DT          -1          Current date and time
FORMAT      -1          Format:  "TUESDAY, APRIL 16, 1974  16:33:32"
```

DT ← IDTIM("*DATE!TIME*")

DATE!TIME is the date and time as a string, in some reasonable format. The internal TENEX representation is returned. TENEX error codes are returned in *!skip!*, which is 0 if no errors occurred.

RUNNING!TIME ← RUNTM(*FORK,@CONSOLE!TIME*)

The running time in milliseconds for *FORK* is returned, and the console connect time is returned in *CONSOLE!TIME*.

DT ← GTAD

The current date and time, in TENEX representation, is returned.

JOBNO ← GJINF(*@LOGDIR,@CONDIR,@TTYNO*)

The user's job number is returned as the value of the call. Reference values are: the number of the logged directory (*LOGDIR*), the connected directory (*CONDIR*), and the TENEX Teletype number (*TTYNO*).

DIRECTORY!NUMBER ← STDIR("*DIRECTORY*",*DORECOGNITION*)

Returns the directory number associated with a string. Any problems are returned in *!skip!* with the code:

1 string does not match
2 string is ambiguous.

Note that "DIRECTORY" MUST BE IN UPPERCASE FOR THE STDIR JSYS.

"*DIRECTORY*" ← DIRST(*DIRNO*)

Returns the string name for directory *DIRNO*. Any problems cause *!skip!* to be set TRUE.

RUNPRG("*PROGRAM*",*INCREMENT*, *NEWFORK*)

This does two entirely different things depending on whether *NEWFORK* is true or not.

If *NEWFORK* is true, then a new fork is created, capabilities are transmitted, and *PROGRAM* is run in the new fork (with the current fork suspended by a WFORK). *INCREMENT* is added to the entry vector location.

If *NEWFORK* is false, then the current fork is replaced with *PROGRAM*. In this case, *RUNPRG* is like the DEC RUN uuo, and hence if the *INCREMENT* is 1, the program is started at the CCL address. If the routine returns at all, there was a problem with the file.

Remember to say .SAV as the *PROGRAM* extension.

10 Pseudo-Interrupts

The interrupt routines in SAIL are closely associated with the special interrupt facility available at the Stanford Artificial Intelligence Project. In order to account for the features needed for the SAIL language (such as the existence of processes within SAIL), and to provide for the use of the TENEX pseudo-interrupt system (PSI), the following decisions have been made:

(1) The interrupt channel numbers will refer to PSI channels, not to channels in the SU-AI or DEC systems. See [2] for a list of these TENEX channels.

(2) Only those interrupt functions that refer to the current TENEX fork will in general be provided.

(3) The TENEX user may use the PSI system independently of the routines described

herein. Thus, machine code functions will generally work together with the functions described here.

(4) The use of the clock interrupt at SU-AI will be emulated by providing functions that initiate interrupts from an inferior fork.

10.1 Immediate and Deferred Interrupts

SAIL provides two kinds of interrupts: immediate and deferred. Immediate interrupts are executed immediately. Deferred interrupts are qued into a list of requests that are executed the next time a polling point is encountered in the program. It should be remarked that a SAIL deferred interrupt is different than a deferred interrupt in the sense of the STIW JSYS. The main point of the SAIL deferred interrupt is to assure that the state of the runtime system is sensible when the code associated with the interrupt is executed.

Immediate interrupts are liberation from SAIL, and the user is warned accordingly. In particular, string operations will cause horrible results if the immediate interrupt happens during string garbage collection.

The following routines are the main TENEX SAIL interrupt routines for immediate interrupt:

PSIMAP	sets up the book-keeping for a PSI channel
ENABLE	turns on a PSI channel
DISABLE	turns off a PSI channel
ATI	associates a control-character with a channel.
DTI	turns off a given control-character

The following example shows the use of thee routines to set up a PSI interrupt. The program is supposed to print out dots forever, but print "HI USER" when a control-Q is typed.

```
BEGIN

SIMPLE PROCEDURE FOO;
OUTSTR("HI USER
");
                           COMMENT note that only constant strings
                           are accessed by FOO.;
PSIMAP(1,FOO,0,3);
                           COMMENT  sets up a level-3 PSI on channel
                           1, and passes the address of FOO to be
                           executed at interrupt level.;
ENABLE(1);                 COMMENT turn on channel 1.;
ATI(1,"Q"-'100);
                           COMMENT associates control-Q with channel 1;
COMMENT body of program.;
DO OUTCHR(".") UNTIL FALSE;
END;
```

WARNING: Immediate interrupts are asynchronous as far as the SAIL runtime system is concerned. While the interrupt handler saves and restores all accumulators, and sets up new interrupt-level stacks, many things may not work transparently. In particular, the creation and access of SAIL dynamic strings is to be avoided, since interrupting during garbage collection will leave the strings in a nonsensical state.

10.2 Deferred Interrupts

Deferred interrupts are safer than immediate ones, although somewhat more difficult to use. The following things complicate deferred interrupts beyond the above example of an immediate interrupt.

(1) Deferred interrupts are handled by a special SAIL process called INTPRO. INTPRO must be sprouted by the process system. This is done by the routine INTSET.

(2) The procedure DFRINT must be passed to PSIMAP. DFRINT is executed at interrupt level, and it buffers the request for the deferred code.

(3) The request for the deferred code is made by creating a special block of code, called a deferred interrupt calling block, and an AOBJN pointer to this block is passed as the third argument to PSIMAP.

(4) Finally, the deferred interrupts must be executed. This is done by including POLL statements in the user's program. Since the deferred interrupts are handled by the special process INTPRO, POLL works just as it would for any other process, with the exception that INTPRO has the highest priority. POLL statements may be inserted by the REQUIRE POLLING!INTERVAL statement; see [3].

The following program illustrates how these things can be accomplished. It is similar in operation to the above immediate interrupt program.

```
BEGIN
PROCEDURE FOO(INTEGER I,J);
OUTSTR("HI " & CVS(I) & " " & CVS(J) & "
");

INTEGER ARRAY FOOBLK[1:4];

FOOBLK[1] ← 4;            COMMENT number of words involved;
FOOBLK[2] ← 12;           COMMENT number of arguments;
FOOBLK[3] ← 13;
FOOBLK[4] ← -1 LSH 18 + LOCATION(FOO);
                          COMMENT address of FOO;

INTSET(NEW,0);            COMMENT NEW returns a new item for
                                  the process INTPRO;
PSIMAP(1,DFRINT,-4 LSH 18 + LOCATION(FOOBLK[1]),3);
ENABLE(1);
ATI(1,"Q"-'100);

DO
    BEGIN
        OUTCHR(".");
        POLL;
    END UNTIL FALSE;

END;
```

Now, whenever a control-Q is typed, DFRINT buffers the request, and makes INTPRO ready to run; then DFRINT DEBRKs (in the sense of the DEBRK JSYS) back to the interrupted code. At SAIL user level, the POLL statement causes the process scheduler to run INTPRO, where the deferred interrupt calling block (which was copied by DFRINT) is used to call FOO. The dots are interrupted by "HI 12 13" being printed out.

10.3 Clock Interrupts

A feature of the SU-AI interrupt system is to interrupt on clock ticks (every 1/60th of a second.) This interrupt happens every tick, regardless of whether the user is the running job or not. SAIL uses the clock interrupt to drive the process machinery via the CLKMOD routine. This feature is not available on ordinary TENEX systems. (IMSSS has implemented a clock interrupt but BBN has declined to support it.)

In order to facilitate the use of processes and to provide for time interrupts on standard TENEX installations, TENEX SAIL has the following three functions:

```
PSIDISMS(PSICHAN,MSTIME)        interrupt every MSTIME milliseconds
PSIRUNTM(PSICHAN,MSTIME)        interrupt every MSTIME milliseconds of runtime
KPSITIME(PSICHAN)               turn off timing
```

Several channels can be interrupted in this way, with different timing intervals. Currently, this is implemented by creating an inferior fork that dismisses, then initiates the appropriate interrupt. These programs run in the accumulators of the inferior fork. The following is an approximation of the program in the inferior fork for PSIDISMS.

```
WAIT:   MOVE    1,[MSTIME]              ;TIME TO DISMISS FOR
        DISMS                           ;GO AWAY
        MOVEI   1,-1                    ;HANDLE TO SUPERIOR FORK
        MOVE    2,[psichan bit mask]    ;SELECTED CHANNEL
        IIC                             ;CAUSE AN INTERRUPT
        JRST    WAIT                    ;CONTINUE
```

The following is an approximation of the program that runs in the inferior fork for PSIRUNTM.

```
WAIT:   MOVEI   1,MSTIME                ;DISMS FOR THIS LONG
        DISMS
        MOVEI   1,-1                    ;SUPERIOR FORK
        RUNTM                           ;GET TOTAL RUNTIME OF SUPERIOR
        CAMGE   1,NEXTTIME#             ;READY
          JRST WAIT                     ;NO
        ADDI    1,MSTIME
        MOVEM   1,NEXTTIME              ;SAVE NEXT TIME TO BE INTERRUPTED
        MOVEI   1,-1                    ;SUPERIOR
        MOVE    2,[psichan bit mask]    ;SELECTED CHANNEL
        IIC                             ;CAUSE INTERRUPT
        JRST WAIT                       ;CONTINUE
```

Note that several different events may cause an interrupt on a given channel. For example, one or more control-characters may be assigned with ATI to a channel that is also being interrupted via a timing fork.

The following causes the SAIL process scheduler to schedule on clock interrupts, using PSI channel number 1.

```
PSIMAP(1,CLKMOD,0,3);
ENABLE(1);
PSIDISMS(1,1000/60);            COMMENT 1/60 of a second --
                                closest to the way this works at SU-AI;
```

10.4 *TENEX Implementation*

The following describes how TENEX SAIL handles its interrupts. The SAIL initialization does a SIR, setting up the tables to external integers LEVTAB and CHNTAB, then does an EIR to turn on the interrupt system. PSIMAP fills the appropriate CHNTAB location with XWD LEV,LEVROU, where LEVROU is the address of the routine that handles the interrupts for level LEV. LEVROU saves the accumulators in blocks PS1ACS, PS2ACS, and PS3ACS, which are external integers, for levels 1 through 3 respectively. Thus, for a level 3 interrupt, accumulator x can be accessed by the expression:

```
MEMORY[LOCATION(PS3ACS) + x]
```

where PS3ACS is an external integer. The PC can be obtained by reading the LEVTAB address with the RIR JSYS, as described in [2].

ENABLE and DISABLE do AIC and DIC JSYSes respectively, and ATI and DTI do the ATI and DTI JSYSes. The user can use such routines as RTIW and STIW to access the interrupts as described in the JSYS manual.

10.5 Interrupt Functions

The following describes the routines that are available. Note that PSICHAN always refers to a TENEX pseudo-interrupt function.

INTMAP(*PSICHAN, ROUTINE, CALLING!BLOCK*)

This is the same function as described in the SAIL manual [3]. It is equivalent to calling PSIMAP with the fourth argument a 3, for interrupt level 3. PSIMAP was added to allow the TENEX SAIL user the ability to use all three levels.

PSIMAP(*PSICHAN, ROUTINE, CALLING!BLOCK, LEVEL*)

PSICHAN is a number of a TENEX pseudo-interrupt channel. *ROUTINE* is executed at interrupt level *LEVEL*. If the interrupt is deferred, then *CALLING!BLOCK* is an AOBJN pointer to the block that says what to do when the deferred code is actually processed later, as more fully described in [3], under the function INTMAP. PSIMAP is like INTMAP except that the *LEVEL* argument has been added.

INTSET(*PROCESS!ITEM, ARGUMENT*)

Sprouts the INTPRO process for deferred interrupts, as in [3].

ENABLE(*PSICHAN*)

Turns on PSICHAN with the AIC JSYS.

DISABLE(*PSICHAN*)

Turns off PSICHAN with the DIC JSYS.

ATI(*PSICHAN, CODE*)

Does the ATI JSYS, associating *CODE* with *PSICHAN*. *CODE* is not necessarily an ASCII character code, but it always is for control-A through control-Z. See [2].

DTI(*CODE*)

Does the DTI JSYS for *CODE*.

DFRINT

The deferring routine, executed at interrupt level. See [3].

DFRINI

See [3].

INTTBL(*SIZE*)

See [3].

CLKMOD

See [3].

PSIDISMS(*PSICHAN*, *MSTIME*)

As described above, creates an inferior fork that initiates an interrupt on *PSICHAN* every *MSTIME* milliseconds.

PSIRUNTM(*PSICHAN*, *MSTIME*)

As described above, creates an inferior fork that initiates an interrupt on *PSICHAN* every time the current fork accumulates *MSTIME* milliseconds of runtime.

KPSITIME(*PSICHAN*)

Turns off any timing interrupt on *PSICHAN* that may have been set up by PSITIME.

***AC1* ← RTIW(*PSICHAN*, @*AC2*)**

Does the RTIW JSYS on *PSICHAN*. Accumulator 1 is the value of the call, accumulator 2 is returned in *AC2*.

STIW(*PSICHAN*, *AC1*, *AC2*)

Does the STIW JSYS on *PSICHAN*. *AC1* and *AC2* go into accumulators 2 and 3.

***STATUS* ← GTRPW(*FORK*)**

The trap status of *FORK* is returned, using the GTRPW JSYS.

The TENEX SAIL system is produced by a fairly complicated process, starting with many files and bootstraps. The entire process is detailed in TELLEM, the SAIL implementer's guide.

It should not however be necessary to go through this process since a complete set of standard TENEX .SAV and .REL files for a recent version will be available from the Stanford Artificial Intelligence Project, either on tape or over the ARPA network. The version currently available will run on TENEX version 1.31.

A complete TENEX SAIL system consists of the following files, listed by the directories on which they should reside. Note that a directory <SAIL> is required. Currently the total size is about 150 pages of disk space.

```
                 <SUBSYS>

SAIL.SAV                        SSAVEd compiler
LOWTSA.REL                      loader bootstrap
                                goes on whichever directory
                                compatibility believes to be SYS:

                 <SAIL>

T-n-SAISGm.SAV                  third segment, where m.n is the
                                version and subversion number
3OPS3.OPS                       START!CODE opcode table, created
                                by MAKTAB.TNX
HLBSAm.REL                      library
LIBSAm.REL                      library
BKTBL.BKT                       standard break table file
UDDT.SAV                        version of UDDT for SAIL
                                includes single-stepping
                                instructions
```

It should be straightforward to obtain and install these files without any reassembly. The sources for TENEX SAIL are available from Stanford or the author.

11.1 Compile-Time Core Map

At compile time, there will usually be four segments in core. The impure data starts at 140; the SAIL compiler starts at about 400000 and ends at about 450000; the opcode table, if needed, runs from about 600000 to 604000; the runtime system runs from 640000 to 670000. In addition, UDDT runs at its customary 770000 to about 775000.

11.2 Runtime Core Map

At runtime for a SAIL-compiled program, there are generally three segments: impure data starting at 140; program and pure data starting at 400000; the runtime system between 640000 and 670000. In addition, pages 600 through 637 are reserved for buffers by the SAIL I/O system. In some cases these buffers will be PMAPped pages of DSK files. Additionally, UDDT, if loaded, will be between 770000 and 775000.

List of Runtime Routines

"" indicates routines named after a jsys.*
"$" indicates routines from DEC SAIL.

References

1. *PDP-10 Reference Manual*, Digital Equipment Corporation, Maynard Mass.

2. *TENEX JSYS Manual*, Bolt, Beranek and Newman Inc., Cambridge, Mass., September, 1973.

3. Van Lehn, Kurt, *SAIL USER MANUAL*, Stanford Artificial Intelligence Laboratory, AIM-204, July, 1973.