```
< NINE, IOEXEC.NLS;28, >, 14-May-78 13:40 KIRK ;;;;

FILE ioexec   % <ARCSUBSYS>XL10  <RELNINE>ioexec %  %
(arcsubsys,xL10,) (RELNINE,ioexec.rel,) %

%universal routines and declarations -- this statement is referenced in
an INCLUDE statement in the file 'nswioexec'%                          2
   ALLOW!
   % error handling %

      %
      Procedures that call err if anything wrong
         open, close, writefile, copfil, openall, closeall, rawopen
      Procedures that can return FALSE with err message
         sysopen, sysclose, opnfil, lgetjfn, sgtjfn, openpc, delpc,
         clsfil, rdhdr
      Procedures that can return warning message
         rdhdr, opnfil, delpc
      %

   %.....Declarations.....%
      REF tda;

   %.....file opening.....%

      (open)   PROCEDURE % Open file routine.  If the value of jfn
      passed to this procedure is zero, it tries to get a jfn (using
      short get jfn with the file name contained in the string whose
      address is passed as the second parameter.)  If it cannot get the
      jfn, err is called.  The full file name is put into astrng.  If
      not already open, the original file whose jfn is passed or has
      been obtained is opened along with its partial copy (if it
      exists); the file is entered in the file status table and the NLS
      file number returned.  If the file is already open, the old file
      number is returned.  If the file cannot be successfully opened,
      the file is removed from the file status table and err is called
      %                                                               2D1
         (jfn,  % jfn of the original file to be opened; if zero,
            we use the file name contained in astrng as parameter to
            sgtjfn to obtain a jfn %
         astrng % Address of string containing file name (if jfn
            zero); may be empty if jfn is a positive number.  In both
            cases the full file name will be placed in the string by
            jfnstr. %
         );
         %----------------%
         LOCAL fileno, fl, flgbits;
         LOCAL STRING errs[150];
         REF fl, astrng;
         %get full file name%
         IF NOT jfn THEN
            BEGIN
            flgbits _ getgtjflg(write, origff, oldvrsn);
               %gets highest extant version or creates version 1%
            IF NOT jfn _ sgtjfn(flgbits, &astrng, $errs) THEN
               err($errs);
```

```
        END;
    jfnstr(jfn, &astrng);
    IF (fileno _ filnum(&astrng)) IN [1,filcnt] THEN
        BEGIN %already open--check validity%
        &fl _ flntadr(fileno);
        IF fl.florig # jfn THEN
            BEGIN
            R1 _ jfn;
            IF NOT SKIP #JSYS closf THEN reljfn(jfn);
            jfn _ fl.florig;
            END;
        IF (fl.flpart OR fl.florig) AND
            fl.flhead THEN
                RETURN(fileno, FALSE) % old file return %
        ELSE delfil(fileno);
        END;
    fileno _ addfil(jfn, &astrng);
    IF NOT opnfil(fileno, $errs) THEN
        BEGIN
        delfil(fileno);
        err($errs);
        END;
    IF errs.L > empty THEN
        BEGIN
        dismes (2, $errs);
        END;
    RETURN(fileno, TRUE);  % new file %
    END.

(openwk) PROCEDURE % Opens and resets work file specified by
astr: creates it with a default extension of .NLS if it does not
exist.  It returns the stid of the origin statement of the file %
                                                            2D2
    (jfn,  % Any value passed here is ultimately not used!!!
        (The space is passed to opwknr which then does a lgetjfn
        anyway %
    astr   % The address of a string containing the name of the
        work file to be opened. %
    );
    LOCAL fileno, stid;
    REF astr;
    % open work file without resetting file without resetting. %
        fileno _ opwknr(jfn, &astr);
    % reset file%
        resetf(fileno);
    % get stid of origin %
        stid _ 0;  %to clear all fields%
        stid.stfile _ fileno;
        stid.stpsid _ origin;
    RETURN(stid)
    END.

(opwknr) PROCEDURE % Opens (or creates) without resetting work
file specified in astr.  Returns the NLS fileno.  The file cannot
be already open!  (opwk handles this case; the two should be
combined.) %
                                                            2D3
```

```
    (jfn,  % The space is used, but the value passed is not!!! %
    astr   % The address of string containing the name of the
       work file to be opened. %
    );
    LOCAL fileno, stid, fl;
    LOCAL STRING fns[50], errsr[150];
    REF astr, fl;
    *fns* _ *astr*; %get some working room%
    IF NOT (jfn _ lgetjfn (0, $frs, $nlsext, getgtjflg (write,
    origff, oldvrsn), $errsr)) THEN err ($errsr);
    jfnstr(jfn, $fns);
    fileno _ addfile(jfn, $fns);
    %now do dummy open and close to create it if it does not
    exist%
       IF NOT sysopen(jfn, write, random, $errsr) THEN
       err($errsr);
       IF NOT sysclose(jfn .V 4B11, $errsr) THEN err($errsr);
    IF NOT opnfil(fileno, $errsr) THEN err($errsr);
    IF errsr.L > empty THEN
       BEGIN
       dismes (2, $errsr);
       END;
    RETURN(fileno)
    END.

(opwk)   PROCEDURE % The same as opwknr, but permits a file to
already be open!  (accomplished through code which exists in
open.  Should be combined with opwknr with parameter added to
check if this code need be executed.) %                          2D4
    (jfn,  % The space is used, but the value passed is not!!! %
    astr   % The address of string containing the name of the
       work file to be opened. %
    );
    LOCAL fileno, stid, fl;
    LOCAL STRING fns[50], errsr[150];
    REF astr, fl;
    *fns* _ *astr*; %get some working room%
    IF NOT (jfn _ lgetjfn (0, $fns, $nlsext, getgtjflg (write,
    origff, oldvrsn), $errsr)) THEN err ($errsr);
    jfnstr(jfn, $fns);
    IF (fileno _ filnum($fns)) IN [1,filcnt] THEN
       BEGIN %already open--check validity%
       &fl _ flntadr(fileno);
       IF fl.florig # jfn THEN
          BEGIN
          R1 _ jfn;
          IF NOT SKIP !JSYS closf THEN reljfn(jfn);
          jfn _ fl.florig;
          END;
       IF (fl.flpart OR fl.florig) AND
          fl.flhead THEN
             RETURN(fileno)
       ELSE
          BEGIN
          delfil(fileno);
          RETURN(FALSE);
```

```
            END;
         END;
      fileno _ addfile(jfn, $fns);
      %now do dummy open and close to create it if it does not
      exist%
         IF NOT sysopen(jfn, write, random, $errsr) THEN
         err($errsr);
         IF NOT sysclose(jfn .V 4B11, $errsr) THEN err($errsr);
      IF NOT opnfil(fileno, $errsr) THEN err($errsr);
      IF errsr.L > empty THEN
         BEGIN
         dismes (2, $errsr);
         END;
      RETURN(fileno)
      END.

(openlock)  PROCEDURE % Open and lock file indicated by astr
after waiting for it to be free.  (Used primarily for systems
files in Journal, Ident systems, etc.)  (Uses OPLOCK to set and
reset openlock lock flag--  this is the only flag used to channel
locks by everyone using the journal, ident, and measurement
systems(!!!))  Expunges directory in effort to try to get file
opened.  Also does some other wierd things.  This is a very
poorly coded procedure which is used in many important
subsystems.  Returns stid of origin statement of file.  SIGNALS
or calls err if something wrong. %                              2D5
   (jfn, % NOT USED!!!!! %
   astr  % Address of string containing name of file to be
   opened. %
   );
   LOCAL stid,  times, mssflg, expunge, njfn, fileno, fl, jdirnm,
   fnstr REF, straddr, newaddr;
   LOCAL STRING fns[50];
   REF astr, fl;
   stid _ 0;
   &fnstr _ $fns;
   INVOKE (expstring);
   %Now check the lock on the file%
      mssflg _ times _ expunge _ 0;
      LOOP BEGIN
         *fnstr* _ *astr*;
         oplock(1); %mke sure only we are locking a file%
         IF (fileno _ rawopen($fns, 0)) THEN EXIT; %Got it%
         oplock(0);
         IF jdebug THEN BEGIN
            *fnstr* _ EOL, *fnstr*, EOL, *sar*;
            dismes (1, &fnstr);
            END;
         IF NOT expunge THEN
            BEGIN
            expunge _ TRUE;
            %expunge connected directory%
            !JSYS gjinf;
            R1 _ R2;
            !JSYS deldf;
            END;
```

```
        IF (times _ times+1) > 24 THEN  %too long%
           BEGIN
           jolock(); %check for a file error from slinker%
           ABORT(2,$"  File Busy--Try Again Later");
           END;
        IF FALSE %NOT mssflg% THEN
           BEGIN
           *fnstr* _ EOL, "filelocked: ", *astr*, "--Waiting.";
           dismes (1, &fnstr);
           BUMP mssflg;
           END;
        %Now Wait for 2500 ms%
           R1 _ 2500; !JSYS dismS; %1678%
        END;
     %By here, file is ours%
        IF NOT [&fl _ flntadr(fileno)].flpart THEN
           BEGIN %make a partial copy%
           !JSYS gjinf;
           jdirnm _ R2; %Put it in the connected directory for
           this one%
           IF NOT setfdb(jdirnm, cinit, &fl) THEN
              BEGIN
              *lit* _ "Cannot Write File";
              GOTO oplockfail;
              END;
           %File is now locked--get PC name and make pc%
              IF fl.flpcst THEN freestring( fl.flpcst := 0,
              $dspblk);
              fl.flpcst _ cpcnam(fl.flastr, jdirnm);
              IF NOT makepc(&fl, fileno, FALSE, $lit) THEN
                 BEGIN
                 %expunge connected directory%
                    !JSYS gjinf;
                    R1 _ R2;
                    !JSYS deldf;
                 IF NOT makepc(&fl, fileno, FALSE, $lit) THEN
                    BEGIN
                    setfdb(0, 0, &fl); %Unlock it%
                    (oplockfail);                          2D5J3A5C3C
                    *lit* _ "Openlock Fail: ", *fns*, EOL,
                    *lit*;
                    oplock(0); %reset flag to let others lock
                    files%
                    err($lit);
                    END;
                 END;
           END;
     oplock(0); %reset flag to let others lock files%
  IF mssflg THEN dismes(0);
  stid.stfile _ fileno;
  stid.stpsid _ origin;
  DROP (expstring);
  RETURN(stid);

  (expstring) CATCHPHRASE (:straddr, newaddr);                 2D5P
     BEGIN
```

```
        CASE SIGNAL OF
           = stringoverflow :
              CASE straddr OF
                 = &fnstr:
                    BEGIN
                    straddr _ getstring(fnstr.M+100, $dspblk);
                    *[straddr]* _ *fnstr*;
                    IF &fnstr NOT= $fns THEN
                       freestring(&fnstr, $dspblk);
                    RESUME(gothelp, &fnstr _ straddr);
                    END;
                 ENDCASE;
           = changestring :
              CASE straddr OF
                 = &fnstr:
                    &fnstr _ newaddr;
                 ENDCASE;
           = return, = unwind:
              BEGIN
              IF &fnstr NOT= $fns THEN
                 freestring(&fnstr := $fns, $dspblk);
              END;
           ENDCASE;
        DISABLE (expstring);
        CONTINUE;
        END;


     END.
(opnctfile) PROCEDURE % Open initial file (in any directory)
specified by the strings whose addresses are passed in initstr
and userstr. CALLED ONLY IN (JNLDEL, OLDIST).  Return stid of
origin if opened, and FALSE if not.  In addition to opening it,
it also locks it by creating a partial copy.  Enables to permit
the operator through this procedure to get to the initial file of
any user.  Calls rawopen. %                                    2D6
    (initstr,  % Address of string containing initials of user. %
    userstr  % Address of string containing user name %
    );
    % Note that there is a potential timing problem in that we
    don't have control over the file if it is not locked when we
    first open it.  We should fix this when we decide how to fix
    the problem for NLS in general %
    LOCAL pcap, fileno, ctlstid, fl, dirnm, lcinit, updinitf;
    LOCAL STRING fname[50], ucustr[50], errstr[200];
    REF initstr, userstr, fl;
    %Initialise parms, and set up signal%
       fileno _ ctlstid _ updinitf _ 0;
       ctlstid.stpsid _ origin;
       INVOKE (sigcls, rtnfl);
    %Build file name%
       *fname* _ "<, *userstr*, ">, *initstr*, ".NLS";
    %enable wheel/operator%
       pcap _ enablw();
    ctlstid.stfile _ fileno _ rawopen($fname, 0);
    %Disable wheel/operator%
```

```
      IF pcap # -1 THEN disablw(pcap);
   IF fileno = 0 THEN
      BEGIN
      IF sar.L < 200 THEN BEGIN
         *errstr* _ *sar*;
         *sar* _ *fname*, ": ", *errstr*;
         END;
      specttyout(0, $sar); %tell operator about control file
      problem%
      DROP (sigcls);
      RETURN(FALSE);
      END;
   &fl _ flntadr(fileno);
   %get users directory number%
      *ucustr* _ *userstr* , 0;
      astruc($ucustr);
      R2 _ chbptr(0) + $ucustr;
      R1 _ 1;
      !JSYS stdir;
      !JRST opncterr;
      !JRST opncterr;
      dirnm _ R1.RH;
      lcinit _ setcinit(&initstr); %make 5-bit ident%
   IF NOT fl.flpart THEN
      BEGIN %create Partial Copy%
      pcap _ enablw();
      setfdb(dirnm, lcinit, &fl);
      IF pcap # -1 THEN disablw(pcap);
      %File is locked...create PC%
         IF fl.flpcst THEN freestring( fl.flpcst := 0, $dspblk);
         fl.flpcst _ cpcnam(fl.flastr, dirnm);
         IF NOT makepc(&fl, fileno, FALSE, $lit) AND fl.flexis
         THEN
            BEGIN
            pcap _ enablw();
            setfdb(0, 0, &fl);
            (opncterr):                                    2D6P5C4
            IF pcap # -1 THEN disablw(pcap);
            close(fileno:= 0);
            DROP (sigcls);
            RETURN(FALSE);
            END;
      END
   ELSE
      BEGIN %see if locked by this user%
      R1 _ fl.florig;
      R2 _ 1000024B;
      R3 _ $R3;
      !JSYS gtfdb;
      IF R3.lkinit # lcinit OR R3.lkdirn # dirnm THEN updinitf _
      TRUE;
      END;
   DROP (sigcls);
   RETURN(ctlstid, updinitf);

   (rtnfl):                                                2D6U
```

```
    DROP (sigcls);
    RETURN (FALSE);

    (sigcls) CATCHPHRASE;                                          2D6Y
       BEGIN
       DISABLE (sigcls);
       CASE SIGNALTYPE OF
          =aborttype:
             BEGIN
                close(fileno := 0);
                TERMINATE;
             END;
          ENDCASE CONTINUE;
       END;

    END.

(openall)   PROCEDURE;  % Open all files in file status table.
Returns FALSE if cannot open one of the files in the file list--
used when user continues from EXEC after quit (files were closed
but not deleted from filst) %                                      2D7
    %-------------------%
    LOCAL fileno, count, fl;
    REF fl;
    IF filcnt NOT IN [1,filmax] THEN RETURN(FALSE);
    fileno _ 1;
    count _ 0;
    &fl _ $filst;
    DO BEGIN
       IF fl.flexis THEN
          IF NOT opnfil(fileno, $lit) THEN
             delfil(fileno)
          ELSE
             BEGIN
             BUMP count;
             END;
       &fl _ &fl + filstl;
       END
    UNTIL (fileno _ fileno + 1) > filcnt;
    RETURN(count);
    END.

(opnfil) PROCEDURE % Primitive open file routine-- read/write
access.  The original file and the partial copy for the given
file number are opened by this routine.  Does not allow pc to be
opened for read unless sysopen fails on pc, (disk allocation
exceeded, for example).  Returns FALSE if original file cannot be
opened.  See coropnfil. %                                          2D8
    (fileno, % NLS file number of file to be opened. %
    astrng   % Address of string to be used for error messages by
       coropnfil and the procedures it calls; is emptied by
       coropnfil, then filled if error (or other information) is
       passed. %
    );
    %----------------%
    REF astrng;
```

```
    RETURN( coropnfil(fileno, &astrng, readwrite) );
    END.

(coropnfil) PROCEDURE % Primitive open file-- specified access. %
                                                              2D9
    % Open file specified by NLS file number passed as the first
    parameter and its partial copy. The access of the partial
    copy may be forced by specifying read access as the third
    parameter. Astrng is used for error and information strings.
    It is emptied at the beginning of this routine. Error return
    is FALSE. It is the responsibility of the calling procedure
    to check the value and decide upon a course of action. Note
    that there may be messages even if a TRUE return is made
    (something about the status of the PC, for example). Thus it
    is also recommended that the calling procedure check the
    length of the string and display it if necessary. THERE IS NO
    USER INTERACTION (I.E., MESSAGE PRINT OUT) IN THIS PROCEDURE.
    %
    (fileno,  % NLS file number of file to be opened. %
    astrng,   % Address of string to be used for error and
       information. %
    accesz    % Access of the partial copy desired. (read,
       readwrite, or write) %
    );
    %accesz refers to PC and may be read%
    LOCAL fi, flags, version, jobno, dirno, fdbwd, sysmsg;
    LOCAL STRING intstr[5], auxerr[150], fname[70];
    REF astrng, fl;
    acchecked _ FALSE;
    &fl _ flntadr(fileno);
    IF fileno NOT IN [1,filcnt] OR
       filcnt NOT IN [1,filmax] OR
       NOT fl.flexis THEN
          BEGIN
          *astrng* _"NLS system error";
          RETURN(FALSE);
          END;
    %open files%
       %get jfn for original file%
       IF NOT fl.florig THEN
          BEGIN
          *fname* _ *[fl.flastr]*;
          flags _ getgtjflg(read, origff, dfltvrs);
          IF NOT fl.florig _ sgtjfn(flags, $fname, &astrng)
          THEN RETURN(FALSE);
          END;
       %open original file%
       IF NOT sysopen(fl.florig, opnmod, random, &astrng) THEN
          RETURN(FALSE);
       %get jfn for partial copy, if exists; open pc%
       %get job and directory number of this user%
          !JSYS gjinf;
          dirno _ R1; %login directory%
       %construct pc name%
          IF fl.flpcst THEN freestring( fl.flpcst := 0, $dspblk);
          fl.flpcst _ cpcnam(fl.flastr, dirno);
```

```
      %get job and directory number for file lock%
         R1 _ fl.florig;
         R2 _ 1000024B; %24TH (octal) word in the file
         data block%
         R3 _ $P3; %put word in R3%
         !JSYS gtfdb; %get file data block%
      *astrng* _ NULL;
      IF R3.lkinit = cinit AND R3.lkdirn = dirno THEN
         BEGIN
         openpc(fileno, &astrng, accesz);
         END
      ELSE
         BEGIN %perhaps old style initials%
         fdbwd _ R3;
         *intstr* _ NULL;
         trnsint(cinit, $intstr);
         IF fdbwd.lkinit .A 4B6 AND fdbwd.lkdirn = dirno
         AND intstr.L = 3 AND intstr[1].oldint = fdbwd.lkinit
         THEN
            BEGIN
            openpc(fileno, &astrng, accesz);
            END
         ELSE IF  fdbwd.lkdirn # 0 THEN %file locked by someone
         else%
            fl.fllock _ TRUE; %set lock bit in file table%
         END;
   IF NOT rdhdr(fileno, $auxerr) THEN
      %rdhdr writes header address into filst entry for this file
      number--initializes the header if things are not cool%
      BEGIN
      % Append rdhdr mesage to astrng %
      IF auxerr.L THEN *astrng* _ *astrng*, SP, *auxerr*;
      RETURN(FALSE);
      END;
   IF NOT acchecked THEN
      BEGIN
      INVOKE (sigmsg, rtnf2);
      IF NOT vrprvacc (fileno, 0, 0, $initsr) THEN
         err ($"Private file; access denied to you");
      DROP (sigmsg);
      END;
   RETURN(TRUE);

   (rtnf2):                                                  2D9R
   DROP (sigmsg);
   RETURN (FALSE);

   (sigmsg) CATCHPHRASE(:sysmsg);                            2D9V
      BEGIN
      DISABLE (sigmsg);
      CASE SIGNALTYPE OF
         = aborttype:
            BEGIN
               IF sysmsg THEN *astrng* _ *astrng*, SP,
               *[sysmsg]*;
               TERMINATE;
```

```
            END;
         ENDCASE CONTINUE;
         END;

      END.

   (rawopen) PROCEDURE % Primitive open at any cost-- Open the file
   specified in astr, enter it into the file list.  Open the partial
   copy whether the file is locked by another or not.  Used only for
   special subsystem files (journal, etc.)  Return fileno if
   successful, 0 otherwise %                                        2010
      (astr, % address of name string of file to be opened. %
      killpc % If TRUE, may unlock file if anything wrong with PC.
         If FALSE, and cannot open PC, do not unlock file, but call
         err %
      );
      LOCAL fileno, jfn, dirnum, fl, pcflag;
      REF astr, fl;
      fileno _ 0;
      IF (jfn _ sgtjfn(getgtjfg(write, origff, oldvrsn), &astr,
      $sar)) = 0 THEN RETURN(0); %File not there%
      INVOKE (sigdel, rtnf3);
      jfnstr(jfn, &astr);
      LOOP
         BEGIN
         IF (fileno _ filnum(&astr)) IN [1,filcnt] THEN
            BEGIN %already open--check validity%
            &fl _ flntadr(fileno);
            IF (fl.flpart OR fl.florig) AND fl.flhead THEN
               BEGIN
               reljfn(jfn);
               EXIT LOOP
               END
            ELSE delfil(fileno);
            END;
         &fl _ flntadr(fileno _ addfile(jfn, &astr));
         IF NOT sysopen(jfn, read, random, $sar) THEN err($sar);
         EXIT LOOP;
         END;
      INVOKE (sigcls, rtnf3);
      R1 _ fl.florig; R2 _ 1000024B; R3 _ $R3; !JSYS gtfdb;
      IF R3.lkdirn # 0 THEN
         dirnum _ R3.lkdirn
      ELSE
         BEGIN
         !JSYS gjinf;
         dirnum _ R1
         END;
      IF fl.flpcst THEN freestring( fl.flpcst := 0, $dspblk);
      fl.flpcst _ cpcnam(fl.flastr, dirnum);
      IF NOT openpc(fileno, $sar, readwrite) THEN
         IF killpc THEN
            IF maywrt(fileno) THEN
               BEGIN %something wrong with pc%
               %unlock the file%
               R1.LH _ 24B; R1.RH _ jfn;
```

```
            R2 _ 0; R2.lkdirn _ R2.lkinit _ 36M;
            R3 _ 0;
            !JSYS chfdb;
            dismes(1, $"File Unlocked");
            END
         ELSE NULL
      ELSE err($sar);
   rdhdr(fileno, $sar);
   DROP (sigdel);
   DROP (sigcls);
   RETURN(fileno);
   (rtnf3):                                                    2D10V
   DROP (sigdel);
   DROP (sigcls);
   RETURN (FALSE);

   (sigdel) CATCHPHRASE;                                       2D10A@
      BEGIN
      DISABLE (sigdel);
      CASE SIGNALTYPE OF
         =aborttype:
            BEGIN
            IF fileno THEN delfil(fileno := 0)
            ELSE reljfn(jfn);
            TERMINATE;
            END;
      ENDCASE CONTINUE;
      END;

   (sigcls) CATCHPHRASE;                                       2D10AB
      BEGIN
      DISABLE (sigcls);
      CASE SIGNALTYPE OF
         =aborttype:
            BEGIN
            close(fileno);
            TERMINATE;
            END;
      ENDCASE
         BEGIN
         CONTINUE;
         END;
      END;

   END.

(openpc)  PROCEDURE % Opens the PC for the file in fileno if it
exists with access specified in acc (if possible).   Returns True
if file does not have  PC or PC opened ok. Returns False if PC
open fail, astr contains message.  %                           2D11
   (fileno, % Of the orignal file whose PC is to be opened. %
   astr,    % Address of astr in which error (or information)
      string will be placed. %
   acc      % Access desired for PC. %
   );
   LOCAL fl, dirnum;
```

```
    REF astr, fl;
    &fl _ flntadr(fileno);
    R1 _ fl.floriq; R2 _ 1000024B; R3 _ $F3;
    !JSYS gtfdb;
    IF (dirnum _ R3.lkdirn) = 0 THEN RETURN(TRUE);
    IF fl.flpcst = 0 THEN
       BEGIN
       *astr* _ "No PC Name String";
       RETURN(FALSE);
       END;
    %Now get jfn%
       IF NOT fl.flpart _ sgtjfn(getgtjflg(read, chkpcf, dfltvrs),
       fl.flpcst, &astr) THEN
          BEGIN
          *astr* _ "PC Does Not Exist";
          filepart[fileno] _ FALSE;
          RETURN(FALSE)
          END
       ELSE filepart[fileno] _ TRUE;
    %Now try to open it%
       IF acc = read  OR NOT sysopen(fl.flpart, readwrite, random,
       &astr) THEN
          BEGIN
          IF sysopen(fl.flpart, read, random, &astr) THEN
             BEGIN %PC open for read only-- disc allocation
             exceded. etc.%
             fl.flpcread _ TRUE;
             *astr* _ "File read only: ", *astr*;
             END
          ELSE %Cannot open PC at all%
             BEGIN
             reljfn(fl.flpart := 0);
             filepart[fileno] _ FALSE;
             *astr* _ "PC Open Fail", EOL, *astr*;
             END;
          RETURN(FALSE)
          END;
    RETURN(TRUE);
    END.

%.....file closing.....%

   (close)   PROCEDURE % Close original file and the partial copy
   and delete from file status table.  Calls err if error occurs.
   writeout called three times! explicitly, in clsfil, and in
   delfil.  Clean this up. %                                    2E1
      (fileno % fileno of file to be closed %
      );
      %----------------%
      %*****RETURN IF FILENO IS ZERO******%
         IF NOT fileno THEN RETURN;
      % write out pages of file and PC and close them %
         IF NOT writeout(fileno, $lit)
         OR NOT clsfil(fileno, $lit) THEN err($lit);
      delfil(fileno);
      RETURN;
```

```
    END.

(closeu)  PROCEDURE % Close and update file %                    2E2
    (fileno % fileno of file to be closed %
    );
    IF fileno = 0 THEN RETURN;
    INVOKE (sigcls);
    updtfi(fileno, 1, 0);
    %Now delete some past copies%
        R1 _ [flntadr(fileno)].florig;
        R2 _ 3;
        IF NOT SKIP !JSYS delnf THEN NULL;
    close(fileno);
    DROP (sigcls);

    RETURN;

    (sigcls) CATCHPHRASE;                                        2E2L
        BEGIN
        DISABLE (sigcls);
        CASE SIGNALTYPE OF
            =aborttype:
            close (fileno := 0);
        ENDCASE CONTINUE;
        END;

    END.

(sigclose) PROCEDURE % Close file after SIGNAL.  Called in
subsystems with NLS system files;  If journal file, lock journal
system.  Close the file specified by fileno. %                   2E3
    (fileno % fileno of file to be closed %
    );
    %This is a rather specialised procedure which is intended to
    be useful to submodes of NLS which use system-type NLS files.
        It is presumed to be called from a SIGNAL, and takes the
        following action:
            It checks to see if the value in SIGNAL is -5 (badfile)
            or -6(I/O Data Error).
                If it is, then it tries to determine whether or not
                the file involved is the one represented by fileno.
                If it is, a message is typed onto the logging TTY,
                and the Journal is locked (insofar as at the time of
                writing all system files are used by the
                Journal--this should probably be changed later.
            The file is closed if fileno is non-zero.
    %
    IF (SIGNAL = -5) AND (fileno = bfilno) THEN
        BEGIN
        %This is the bad file%
        lockjo(1); %lock Journal%
        *lit* _ "Bad System File = ", *[[flntadr(fileno)].flastr]*;
        specttyout(0, $lit);
        specttyout(308, $lit);
        bfilno _ 0;
        END;
```

```
    IF SIGNAL = -6 THEN
       BEGIN %I/O Data Error--see if this is the file%
       %Code goes here to figure out which file it was%
       END; %For now, simply pass it on%
    close(fileno);
    RETURN;
    END.

(closeall) PROCEDURE; % Close all files in file status table. %
                                                                  2E4
    %--------------------%
    LOCAL fileno, fl;
    REF fl;
    IF filcnt NOT IN [0,filmax] THEN rerror();
    fileno _ 0;
    &fl _ $filst;
    UNTIL (fileno _ fileno + 1) > filcnt DO
       BEGIN
       IF fl.flexis AND NOT clsfil(fileno, $lit) THEN
          dismes(1, $lit);
       &fl _ &fl + filstl;
       END;
    RETURN;
    END.

(clsfil)  PROCEDURE  % Primitive close file routine.  Does not
delete file from file status table, but does do writeout.  The
original file and the partial copy are closed by this routine.
If anything goes wrong, it returns false with error message in
astrng.  Close, closeu, closeall, and clsfil should be examined
for possible combination. %                                       2E5
    (fileno, % File number of file to be closed. %
    astrng   % Address of string to be used for error message. %
    );
    %----------------%
    LOCAL fl;
    REF fl, astrng;
    IF fileno NOT IN [1,filcnt] OR filcnt NOT IN [1,filmax]
    THEN
       BEGIN
       *astrng* _ "NLS system error";
       RETURN(FALSE);
       END;
    &fl _ flntadr(fileno);
    IF NOT writeout(fileno, &astrng) THEN
       RETURN(FALSE);
    IF fl.flpart THEN
       BEGIN
       IF NOT sysclose(fl.flpart, &astrng) THEN
          RETURN(FALSE);
       fl.flpcread _ fl.flpart _ 0;
       filepart[fileno] _ FALSE;
       END;
    IF fl.florig AND NOT sysclose(fl.florig, &astrng) THEN
       RETURN(FALSE);
    fl.florig _ 0;
```

```
    RETURN(TRUE);
    END.

(closepc) PROCEDURE % Closes the PC for the file in fileno if it
exists. (Currently used only in ident system.) %                    2E6
    (fileno % of file whose PC is to be closed %
    );
    LOCAL fl, dirnum;
    LOCAL STRING work[100];
    REF astr, fl;
    &fl _ fintadr(fileno);
    IF fl.flpart THEN
        BEGIN
        IF NOT writeout(fileno, $work) THEN err($work);
        IF NOT sysclose(fl.flpart, $work) THEN
            reljfn(fl.flpart);
        fl.flpcread _ fl.flpart _ 0;
        filepart[fileno] _ FALSE;
        IF NOT rdhdr(fileno, $work) THEN err($work);
        END;
    RETURN;
    END.

%.....null file, file reset, and initial file routines.....%

(openid) PROCEDURE;  % open initial file; constructs file name
from login directory name and current initials. creates the file
if it does not exist.  Returns file number. %                       2F1
    %----------------%
    LOCAL jfn, flgbits, word;
    LOCAL STRING dname[30], nfname[50];
    %build file name%
    %get login directory name%
    gdname(cuno, $dname);
    *nfname* _ "<, *dname*, ">, *initsr*, ".NLS";
    flgbits _ getgtjflg(write, origff, oldvrsn);
        %gets highest extant version or creates version 1%
    IF NOT jfn _ sgtjfn(flgbits, $nfname, $lit) THEN err($lit);
    INVOKE (sigoid, rtnopen);
    % set up last read date of initial file %
        !gtfdb ( jfn, 186+$fdbref, $lstred);
    (rtnopen):                                                      2F1L
    RETURN(open(jfn, $nfname));

    (sigoid) CATCHPHRASE;                                           2F10
        CASE SIGNAL OF
            =errsig:  % open has failed and called err %
                BEGIN
                DISABLE (sigoid);  % no longer catch signals %
                % open with write and close to create the file %
                    % if open fails, jfn will be released, must get it
                    back %
                % will be initialized by rdhdr %
                IF NOT jfn _ sgtjfn(flgbits, $nfname, $lit) THEN
                err($lit);
                word _ 0;
```

```
            word.fdbnar _ TRUE;   %say dont archive%
            chnfdb(jfn, 17B, word, word); % set file don't
            archive %
            IF NOT sysopen(jfn, write, random, $lit) OR
            NOT sysclose(jfn .V 4B11, $lit) THEN err($lit);
            % set up last read date of initial file %
               !gtfdb ( jfn, 1B6+$fdbref, $lstred);
            TERMINATE;
            END;
         ENDCASE
            CONTINUE;

   END.

(openul) PROCEDURE  % create file with given name.  Returns file
number. (Almost identical with openid except for name creation
and flags sent to sgtjfn; they should be combined (add flag
parameter).) %                                                    2F2
   (nfname % Address of name of file to be opened. %
   );
   %----------------%
   LOCAL jfn;
   IF NOT jfn _ sgtjfn(gtjoof, nfname, $lit) THEN err($lit);
   INVOKE (sigonull, rtnop2);
   (rtnop2):                                                      2F2G
   RETURN( open(jfn, nfname) );

   (sigonull) CATCHPHRASE;                                        2F2J
   BEGIN
   DISABLE (sigonull);  % no longer catch signals %
   CASE SIGNAL OF
      =errsig:  % open has failed and called err %
         BEGIN
         % open with write and close to create the file %
            % if open fails, jfn will be released, must get it
            back %
         % will be initialized by rdhdr %
         IF NOT jfn _ sgtjfn(gtjoof, nfname, $lit) THEN
         err($lit);
         IF NOT sysopen(jfn, write, random, $lit) OR
         NOT sysclose(jfn .V 4B11, $lit) THEN err($lit);
         TERMINATE;
         END;
      ENDCASE
         CONTINUE;
   END;

   END.

(resetf) PROCEDURE % Reset file-- null partial file%              2F3
   (fileno % NLS file number of file to be reset. %
   );
   % This routine is called to convert the file whose number
   is passed to a null file.  The original file is unchanged
   but the PC is zapped.%
   %----------------%
```

```
    IF fileno NOT IN [1, filcnt] THEN err($"NLS system error");
    IF NOT writeout(fileno, $lit)
    OR NOT delpc(fileno, $lit) THEN err($lit);
    IF lit.L > empty THEN
       BEGIN
       dismes(2, $lit);
       END;
    intfil(fileno);
    RETURN;
    END.

%.....file initialization.....%

  (intfil) PROCEDURE  % Initialize file-- initialize header and
  create dummy origin statement %                                  2G1
     (fileno % NLS file number of file being initialized %
     );
     %-----------------%
     LOCAL stid, ring;
     LOCAL TEXT POINTER orgpoint;
     REF ring;
     % Initialize file header, get origin ring & stid %
        setfil(fileno);
        stid _ newrng(fileno : &ring);
     % Check validity-- should be origin %
        IF stid.stpsid NOT= origin THEN err($"NLS system error");
     % Fix up ring flags for origin %
        ring.rhf _ TRUE;
        orgpoint _ orgpoint[1] _ 0;
        orgpoint.stpsid _ ring.rsuc _ ring.rsub _ origin;
        orgpoint.stfile _ fileno;
     % Set text of origin statement %
        gdmys($orgpoint);
     RETURN;
     END.

  (setfil) PROCEDURE  % Initialize file header and core page status
  table.  The header of the file whose number is passed is
  initialized by this routine.  All status blocks are set to empty,
  the marker table is set to empty, and the
  CORPST(core-page-status-table) entries pertaining to this file
  (other than the header) are set to indicate that no pages are
  loaded. %                                                        2G2
     (fileno % NLS file number of file being initialized %
     );
     %--------------------%
     LOCAL  index, header, pgindex, fb, fl, block;
     REF fb, fl;
     %initialize file header by updating dummy file header
     and copying that to the real file header%
        nlsvwd _ 1;
        namdl1 _ dfnmdl; %use default name delimiters from
        user-option page%                                          2G2G2
        namdl2 _ dfnmdr;%use default name delimiters from
        user-option page%
        % file owner (user number) %
```

```
        !JSYS gjinf;
        funo _ R2;
    finit _ lwtim _ sidcnt _ 0;
    % creation date %
        fcredt _ gtadcall();  %get time and date%
    mkrtbl _ 0;
    % clear ring and data status tables in header %
        clrngs($rngst);
        cldtbs($dtbst);
&fl _ flntadr(fileno);
IF fl.flhead = 0 THEN
    BEGIN %create a header for it%
    pgindex _ lodrfb(0, -1); %get a block%
    frzblk(pgindex, 1);
    INVOKE (sigrlse);
    corpst[pgindex].ctfile _ fileno;
    fl.flhead _ pgindex;
    %initialize header of new block%
    &fb _ crpgad[pgindex];
    %pmap jsys to make the header page zero of the file%
        % R1 _ JFN in left half, 0 in right half %
        IF fl.flpart THEN
            BEGIN
            R1.LH _ fl.flpart;
            R3 _ 14B10;
            END
        ELSE
            BEGIN
            IF NOT (R1.LH _ fl.florig) THEN
                err($"NLS system error");
            R3 _ 10048B;
            END;
        R1.RH _ 0;
        !rpacs(); %see if page exists%
        IF R3 .A 4B10 OR R2 .A 1B10 THEN
            BEGIN
            R2.LH _ 4B5;
            R2.RH _ &fb / 512;
            !JSYS pmap;
            END;
    fb.fbind _ fb.fbpnum _ 0;
    fb.fbtype _ hdtyp;
    END;
filehead[fileno] _ header _ filhdr(fileno);
mvbfbf($filhed, header, $filhde-$filhed);
%clear core used by file except file header%
    clrcor(fileno);
DROP (sigrlse);
RETURN;

(sigrlse) CATCHPHRASE;                                          2G2P
    BEGIN
    DISABLE (sigrlse);
    CASE SIGNALTYPE OF
        =aborttype:
            frzblk(pgindex, -1);
```

```
        ENDCASE;
        CONTINUE;
        END;

    END.

% build dummy statement %
    (gdmys) PROCEDURE (ptr);                                        2G3A
        LOCAL STRING fname[150], datestr[50];
        REF ptr;
        *fname* _ NULL;
        IF NOT ptr.stastr THEN filnam(ptr.stfile, $fname);
        *datestr* _ NULL;
        <AUXCOD, getdat>($datestr);
        ST ptr _ *fname*, ", ", *datestr*, SP, *initsr*, " ;;;;";
        RETURN;
        END.

%.....system file I/O interaction (JFNs, OPEN/CLOSE JSYS
STUFF).....%

    (sysopen) PROCEDURE % Open jsys-- set up registers for and
    execute openf JSYS.  Return TRUE if successful, FALSE if failure
    with error message in string whose address was passed as fourth
    parameter. %                                                    2H1
        (jfn,  % jfn of file to be opened. %
        accessmode, % Access desired: read, write, readwrite, or
        append %
        type,  % chrtyp, bintyp, random, comtyp, lpttype %
        astrng % Address of string for error message %
        );
        LOCAL STRING locstr[60];
        LOCAL bp, error, syflag, count, cnttim;
        REF astrng;
        FOR cnttim_0 UP
        UNTIL >1 DO
            BEGIN
            %open file%
            R1 _ jfn;
            CASE accessmode OF
                = read: syflag _ 285;
                = write: syflag _ 185;
                = readwrite: syflag _ 385;
                = append: syflag _ 284 + 185;
                = rWthawed: syflag _ 302B3;
                = rthawed: syflag _ 202B3;
                ENDCASE
                    BEGIN
                    *astrng* _ "NLS system error";
                    RETURN(FALSE);
                    END;
            CASE type OF
                = chrtyp: R2 _ syflag .V 7B10; % 7 bit byte %
                = bintyp: R2 _ syflag .V 44B10; % 36 bit byte %
                = random: R2 _ syflag;
                = comtyp: R2 _ syflag .V 7B10; % 7 bit byte %
```

```
        =lpttype: R2 _ syflag .V 7B10; % 7 bit byte %
        = netype: R2 _ syflag .V 1024B8; % 8 bit byte; buffered
        send mode %
        ENDCASE
           BEGIN
           *astrng* _ "NLS system error";
           RETURN(FALSE);
           END;
     R2 _ R2 .V 2B2; %Never wait-- error if file busy.%
     IF NOT SKIP !JSYS openf THEN
        IF NOT cnttim THEN
           BEGIN %dismiss and try again-- TENEX timing problem%
           R1 _ 1000;
           !JSYS disms;
           END
        ELSE
           BEGIN
           error _ R1;
           jfnstr(jfn, $locstr);
           CASE error OF
              = $opnx1: *astrng* _ *locstr*, " already open";
              = $opnx2:
                 IF accessmode=read THEN
                    *astrng* _ *locstr*, " does not exist"
                 ELSE
                    *astrng* _ "You cannot write on ", *locstr*;
              = $opnx9: *astrng* _ *locstr*, " is busy";
              = $opnx10: *astrng* _ "File space allocation
              exceeded";
              ENDCASE *astrng* _ *locstr*, " cannot be opened";
           RETURN(FALSE);
           END
     ELSE EXIT;
     END;
  %set byte size for write files until tenex bug fixed%
  IF (accessmode = write) AND (type # random) AND (type #
  lpttype) AND (type # netype) THEN
     BEGIN % set byte size (again) %
     CASE type OF
        =  chrtyp: R3 _ 7B8;
        =  bintyp: R3 _ 44B8;
        = comtyp: R3 _ 7B8;
        = lpttype: R3 _ 7B8;
        ENDCASE
           BEGIN
           *astrng* _ "NLS system error";
           RETURN(FALSE);
           END;
     R1.LH _ 11B;  %offset to word to be changed%
     R1.RH _ jfn;
     R2 _ 77B8;  %bits to be changed%
     !JSYS chfdb;
     END;
  RETURN(TRUE);
  END.
```

```
(sysclose) PROCEDURE % Issue close jsys-- Return TRUE if file
closed, FALSE if not with error message in string whose address
was passed.%                                                    2H2
   (jfn,  % Of file to be closed %
   astrng % Address of string for error message %
   );
   LOCAL bp, error, count;
   LOCAL STRING locstr[60];
   REF astrng;
   %close file%
      R1 _ jfn;
      IF NOT SKIP !JSYS closf THEN
         BEGIN
         jfnstr(jfn, $locstr);
         *astrng* _ *locstr*, "cannot be closed";
         RETURN(FALSE);
         END;
   RETURN(TRUE);
   END.

%lgetjfn and sgtjfn ar in 'NSW affected routines' branch%
(reljfn) PROCEDURE % release the jfn %                          2H4
   (jfn % to be released %
   );
   R1 _ jfn;
   IF NOT SKIP !JSYS rljfn THEN !JFCL 0;
   RETURN;
   END.

(chtdbjsys)       % call CHFDB %
PROCEDURE(dsplc, jfn, mask, newbits);                           2H5
   LOCAL tmpac1;
   tmpac1.LH _ dsplc;
   tmpac1.RH _ jfn;
   !chfdb( tmpac1, mask, newbits );
   RETURN;
   END.

(sysdelete)       % system delete file %
PROCEDURE( jfn );                                               2H6
   RETURN( IF NOT SKIP !delf( jfn ) THEN FALSE ELSE TRUE );
   END.

(gtjerr) PROCEDURE % Convert error word from gtjfn into string %
                                                                2H7
   (error, % Code returned by unsuccessful gtjfn %
   string, % Address of string for error message %
   flags   % Passed to gtjfn (used to check access) %
   );
   REF string;
   *string* _ *[ CASE error OF
      =$gjfx4: $"Illegal character in file name";
      IN [$gjfx5, $gjfx14], =$gjfx25:  $"Illegal file name";
      =$gjfx16:  $"No such device";
      =$gjfx17:  $"No such directory";
      =$gjfx18:  $"No such file name";
```

```
        =$gjfx19:  $"No such extension";
        =$gjfx20: $"No such version";
        =$gjfx21: $"No such file";
        =$gjfx22: $"Can't reference any more files";
        =$gjfx23: $"No room in directory";
        =$gjfx24:
           IF flags.LH .A gtjwrt THEN
              $"You cannot create a new file in this directory"
           ELSE
              $"Filename not recognized;
              may be misspelled, in another directory, or
              archived.";
        =$gjfx32: $"No files match this specification";
        ENDCASE $"System file error" ]*;
     RETURN;
     END.

(getgtjflg) PROCEDURE %get flags for gtjfn JSYS%                    2H8
     (accessmode,  % Access desired for file %
     chkpc,  % If chkpcf (=TRUE), this is for a PC: No access by
        other forks.  If orgiff (=FALSE), this is an original file:
        access by other forks permitted. %
     version % for right half of flagword (see JSYS manual).
        Oldversion (-4) treated differently.: no longer permitted
        turn off B0, put 0 (highest existing verion ) in RH. %
     );
     LOCAL gtflag;
     CASE version OF
        = oldvrsn:  % Old -4 right half; B0 must be off, RH 0 now %
           BEGIN
           % LH; B0 must be off. %
           IF chkpc THEN
              gtflag.LH _ gtjcpc % No access by other forks %
           ELSE
              gtflag.LH _
                 (IF accessmode = read THEN gtjred ELSE 0);
           % RH of flag word %
              gtflag.RH _ 0; % Get highest existing version because
              B0 off. %
           END;
        ENDCASE  % defltvrsn, -1, -2, etc.  %
           BEGIN
           % LH %
           IF chkpc THEN
              gtflag.LH _
                 gtjcpc .V (IF accessmode = read THEN gtjred ELSE
                 gtjigdel)
           ELSE
              gtflag.LH _
                 (IF accessmode = read THEN gtjred ELSE gtjwrt);
           % RH of flag word %
              gtflag.RH _ version;
           END;
     RETURN(gtflag);
     END.
```

```
%.....get dir for this savfile.....%
   (dirfrom)   % CL:UL;  puts dir of this save file into string %
   PRLCEDURE (dstr REF % => dstr REF %);                              211
      % Procedure description
         FUNCTION
            This proc puts the name of the directory of the save
            file that this page came from into its argument, an L10
            string.  The page used is the first one found that is
            not private to this fork.
         ARGUMENTS
            dstr REF - address of L10 string
         RESULTS
            dstr REF - address of L10 string
         NON-STANDARD CONTROL
            err is called if no nonprivate page can be found or if
            string is too short.
         GLOBALS
            none
         %
      % Declarations %
         LOCAL CONSTANT
            privpage = 2B8, %page is private%
            exists = 1B10; %page exists in address space%
         LOCAL
            page, %page number of nonprivate page%
            jfn; % job file number of file page came from%
      % find a page that is not private %
         FOR page _ 1 UP UNTIL > 777B DO
            BEGIN
            !rpacs( 4B11 + page); %get page access in this fork%
            IF (R2 .A exists) THEN %page exists, check if private%
               IF NOT (R2 .A privpage) THEN EXIT LOOP; %if not
               private, done%
            END;
         IF page > 777B THEN err($"Can't get page -DIRFROM");
      % get jfn for page %
         !rmap(4B11 + page); %page number in this fork%
         jfn _ R1.LH;
      % get directory for jfn %
         !jfns( &dstr + chbmty, jfn, 1B10);
      % find string length %
         dstr.L _ 1;
         LOOP % looking for zero char %
            CASE *dstr*[dstr.L] OF
               NOT= 0: BUMP dstr.L;
               ENDCASE
                  BEGIN
                  BUMP DOWN dstr.L;
                  EXIT LOOP;
                  END;
         IF dstr.L > dstr.M THEN err($"String too short -DIRFROM");
      %release jfn%
         !rljfn(jfn);
      % Return %
         RETURN( &dstr );
      END.
```

```
%.....file status table management routines.....%

   (addfil) PROCEDURE % Add a file to file status table. Returns NLS
   file number if successful; calls err if global filcnt is invalid,
   abort if too many files are open. %                              2J1
      (jfn,  % of file to be added to filst.   If 0, florig field
      remains unchanged (zero?) %
      astrng % Address of string containing full file name %
      );
      %a new file is added to the file status table.  Astrng
      is the address of a string containing (for now) a full
      file name.  If jfn > 0 then it is stored into florig.%
      %----------------%
      LOCAL fl, unused, fileno, end;
      REF fl, astrng;
      IF filcnt NOT IN [0,filmax] THEN err($"NLS system error");
      unused _ FALSE;
      end _ (&fl _ $filst-filstl) + filcnt*filstl;
      UNTIL (&fl _ &fl + filstl) > end DO
         IF NOT fl.flexis THEN
            BEGIN
            unused _ TRUE;
            EXIT;
            END;
      IF NOT unused THEN
         BEGIN
         IF filcnt = filmax THEN err($"Too many files open");
         &fl _ (filcnt := fileno _ filcnt + 1)*filstl + $filst;
         END
      ELSE fileno _ (&fl - $filst)/filstl + 1;
      %zero the filst entry%
         clrfil(fileno);
      IF fl.flastr THEN freestring( fl.flastr := 0, $dspblk);
      fl.flastr _ getstring( astrng.L + 1, $dspblk);
      *[fl.flastr]* _ *astrng*;
      IF jfn > 0 THEN fl.florig _ jfn;
      fl.flexis _ TRUE;
      BUMP fopncnt[fileno];
         % The file status table entry is being used (again) for a
         new file %
      IF debugaccess THEN fl.flaccm _ access;
      %Get Directory number for file%
         fldirectory(&fl);
      RETURN (fileno);
      END.

   (delfil) PROCEDURE % Delete file (which has been closed) from NLS
   internal bookeeping including file status table,  display area
   table, and other arrays. Write out called here and in several
   places before it (see close)!  Clean this up.  %                 2J2
      (fileno % of file to be deleted. %
      );
      %----------------%
      LOCAL fl, end, da;
      REF fl, da;
```

```
    IF filcnt NOT IN [0, filmax] OR fileno NOT IN [1, filcnt]
    THEN RETURN;
    &fl _ flntadr(fileno);
    %must write a routine that will release all frozen dstatements
    from a particular file, in all display windows -- (relall
    doesn't do it)%
    %relall(fileno);% %release any frozen statements%
    IF NOT writeout(fileno, $lit) THEN err($lit);
    %clean up window array%
    end _ (&da _ $dpyarea) + dal*dacnt;
    UNTIL &da = end DO
       BEGIN
       IF da.daexis AND NOT da.daempty AND
       da.dacsp NOT= endfil AND
       da.dacsp.stfile = fileno THEN
          BEGIN
          da.dacsp _ endfil;
          da.daempty _  TRUE;
          END;
       &da _ &da + dal;
       END;
    %clean up filst entry%
    IF fl.flpart OR fl.florig THEN %error condition -- get rid of
    jfns%
       BEGIN
       IF fl.flpart THEN
          BEGIN
          R1 _ fl.flpart;
          IF NOT SKIP !JSYS closf THEN reljfn(fl.flpart);
          END;
       IF fl.florig THEN
          BEGIN
          R1 _ fl.florig;
          IF NOT SKIP !JSYS closf THEN reljfn(fl.florig);
          END;
       END;
    fl.flpcread _ fl.flpart _ fl.florig _ fl.flhead _ 0;
    filepart[fileno] _ FALSE;
    fl.flexis _ FALSE;
    IF fl.flastr THEN freestring( fl.flastr := 0, $dspblk);
    IF fl.flpcst THEN freestring( fl.flpcst := 0, $dspblk);
    IF fileno = filcnt THEN BUMP DOWN filcnt;
    %take care of swork if necessary%
       IF NOT swork.stastr AND swork.stfile = fileno THEN
          swork _ endfil;
       %similar to <NLS, FILMNP, frerng>.  ensures that fechcl
       won't try to use a bad stid from swork%
    RETURN;
    END.

(cpyfle) PROCEDURE % Copy file status table entry for newfno to
oldfno.  Used in outfile to force "new" file to use "old" file
number and filst. %                                          2J3
   (newfno, % file whose status table entry is to be copied %
   oldfno   % file whose filst entry is to be copied to %
   );
```

```
    LOCAL fl1, fl2;
    REF fl1, fl2;
    &fl1 _ flntadr(newfno);
    &fl2 _ flntadr(oldfno);
    fl2.flexis _ fl1.flexis;
    fl2.flhead _ fl1.flhead;
    fileheadColdfnoC _ fileheadCnewfnoC;
    fl2.flpart _ fl1.flpart;
    fl2.flpcread _ fl1.flpcread;
    filepartColdfnoC _ filepartCnewfnoC;
    fl2.florig _ fl1.florig;
    IF fl2.flastr THEN freestring( fl2.flastr := 0, $dspblk);
    fl2.flastr _ getstring( Cfl1.flastrC.L, $dspblk);
    *Cfl2.flastrC* _ *Cfl1.flastrC*;
    IF fl2.flpcst THEN freestring( fl2.flpcst := 0, $dspblk);
    fl2.flpcst _ getstring( Cfl1.flpcstC.L, $dspblk);
    *Cfl2.flpcstC* _ *Cfl1.flpcstC*;
    fl2.fldirno _ fl1.fldirno;
    RETURN;
    END.

(flntadr) PROCEDURE % Returns the address of the file list entry
for the file number passed %                                    2J4
    (fileno % NLS file number whose filst entry address is sought
    %
    );
    LOCAL fl; REF fl;
    IF fileno NOT IN C1, filcntC THEN
        err($"NLS system error: illegal file # passed to FLNTADR");
    &fl _ (fileno-1)*filstl + $filst;
    %IF NOT fl.flexis THEN
        err($"NLS system error: illegal file # passed to
        FLNTADR");%
    RETURN(&fl);
    END.

(filnam) PROCEDURE % Get file name from filst entry for file
whose file number is passd.  Put into string whose address is
passed %                                                        2J5
    (fileno, % file number whose name string is sought %
    astrng   % address of string into which name is placed %
    );
    %retrieve the file name for the given file number from
    the filest%
    %------------------%
    LOCAL fl, fjfn;
    LOCAL STRING locstrC200C;
    REF fl, astrng;
    &fl _ flntadr(fileno);
    IF (fjfn _ fl.florig) = 0 THEN
        err($"NLS system error: illegal fst entry detected in
        FILNAM");
    jfnflink( fjfn, $locstr, 011110B6+1 );
    *astrng* _ *astrng*, *locstr*;
    RETURN;
    END.
```

```
(filnum) PROCEDURE % Get NLS file number from filst given file
name string.  Returns -1 if error or string not in table. %    2J6
   (astrng %  Address of string containingfull file name to
      be compared with names in filst. %
   );
   %------------------%
   LOCAL fl, end;
   REF fl;
   IF filcnt = 0 OR astrng = -1 THEN RETURN (-1);
   IF filcnt NOT IN [1,filmax] THEN err($"NLS system error");
   end _ (&fl _ $filst) + filcnt*filstl;
   DO IF fl.flexis AND
      compas(fl.flastr, astrng) THEN
         RETURN((&fl-$filst)/filstl+1)
   UNTIL (&fl _ &fl + filstl) >= end;
   RETURN(-1);
   END.

(fldirectory) PROCEDURE % Get directory number given address of
filst entry. %                                                 2J7
   (fl % Address of filst entry. %
   );
   LOCAL TEXT POINTER ptr1, ptr2;
   LOCAL STRING dirs[40];
   REF fl;
   fl.fldirno _ getdno(fl.flastr);
   RETURN;
   END.

(transdir) PROCEDURE % Translates string to directory number
using stdir jsys.  Accepts address of a string, appends 0 to it
as required by TENEX.  If not a match calls err.  Else returns
directory number.%                                             2J8
   (string % Address of directory string; zero will be appended
   (so
      must have room!) %
   );
   REF string;
   *string* _ *string*, 0;
   R1 _ 1; %exact match%
   R2 _ chbptr(empty) + $string;
   !JSYS stdir; %stdir%
   GOTO dirgm;
   GOTO diram;
   RETURN(R1.RH);
   (dirgm):  err($"No such directory");                        2J8K
   (diram): err($"Ambiguous directory specification");         2J8L
   END.

(getenf)  % get EXTERNAL NAME LINK FILE name %                 2J9
   PROCEDURE
      (fn,        % file number for which we want the ENLF name %
      type,       %TRUE, check the file origin; FALSE, get
      user-profile immed%
      astr);      % astr to get the ENLF %
```

```
    LOCAL TEXT POINTER stid, tps, tp1, tp2;
    LOCAL adstr[40];
    REF astr;

    % RETURNS %
       % FALSE: if it cannot find an EXTERNAL NAME LINK FILE name
       TRUE: if it finds an EXTERNAL NAME LINK FILE name
          If type is TRUE, searches the origin statement of the
          file whose file number it is passed for the string
          "EXTERNAL LINKS:" followed by any number of spaces,
          tabs, carriage returns, linefeeds, and or end-of-lines.
          If a link is present here it is considered to be the
          EXTERNAL NAME LINK FILE.  If these conditions are not
          met, then it assumes that the global string enlfstr (in
          user-option page) is the name of the EXTERNAL NAME LINK
          FILE iff enlfstr is NOT null.
       %

    stid _ origin;  stid.stfile _ fn;  stid[1] _ 1;
    IF (fn NOT= endfil.stfile) AND type AND
       FIND SF(stid) ["EXTERNAL LINKS:"] $(SP/TAB/CR/LF/EOL) ^tps
       THEN
          BEGIN
          INVOKE (signoenlf, noenlf);
          lnkprs( $tps, $adstr );
          DROP (signoenlf);
          tp1 _ adstr[ls];  tp1[1] _ adstr[ls+1];
          tp2 _ adstr[le];  tp2[1] _ adstr[le+1];
          IF tp1[1] # tps[1] THEN GOTO noenlf;
          *astr* _ tp1 tp2;
          END
    ELSE
       (noenlf):  BEGIN                                          2J9J1
       IF NOT enlfstr.L THEN RETURN( FALSE );
       *astr* _ *enlfstr*;
       END;
    RETURN( TRUE );

    (signoenlf) CATCHPHRASE;                                     2J9N
       BEGIN
       DISABLE (signoenlf);
       TERMINATE;
       END;

    END.

%.....status table initialization utility procedures.....%

   (clrfil) PROCEDURE %Clear the filst entry for the given fileno.%
                                                                 2K1
       (fileno % file number whose filst entry will be cleared. %
       );
       %----------------%
       LOCAL fl;
       REF fl;
       &fl _ fintadr(fileno);
```

```
    fl.flexis _ FALSE;
    fl.flhead _ fl.flpcread _ fl.flpart _ fl.florig _ fl.fllock _
    0;
    filepart[fileno] _ FALSE;
    filehead[fileno] _ 0;
    IF fl.flastr THEN freestring( fl.flastr := 0, $dspblk);
    IF fl.flpcst THEN freestring( fl.flpcst := 0, $dspblk);
    RETURN;
    END.

(clrngs) PROCEDURE % Clear ring status table; could be combined
with almost identical cldtbs %                                  2K2
    (rn % Address of ring status table in dummy file header %
    );
    LOCAL rngind;
    REF rn;
    rngl _ rngind _ 0;
    DO BEGIN
       rn _ 0;
       BUMP &rn;
       END
    UNTIL (rngind _ rngind+1) = rngm;
    RETURN END.

(cldtbs) PROCEDURE % Clear data status table; could be combined
with almost identical clrngs %                                  2K3
    (dt % Address of data block status table in dummy file header
    %
    );
    LOCAL dtbind;
    REF dt;
    dtbl _ dtbind _ 0;
    DO BEGIN
       dt _ 0;
       BUMP &dt;
       END
    UNTIL (dtbind _ dtbind+1) = dtbm;
    RETURN END.

(clrcor) PROCEDURE  % clear core page status table entries and
remove pages from core pertaining to this file (except for the
header, the address of which will be returned.).  Used in
initialization of file.  Similar to writeout! %                 2K4
    (fileno % number of file whose corpst entries are to be
    cleared;
       if =-1, the whole corpst is to be cleared. %
    );
    LOCAL pgindex, ct, hdindex;
    REF ct;
    pgindex _ 1;
    hdindex _ -1;
    &ct _ $corpst + 1;
    DO BEGIN
       IF ct.ctfull AND
          (fileno = -1 OR fileno = ct.ctfile) THEN              2K4H1A
             IF ct.ctpnum = 0 THEN hdindex _ pgindex
```

```
                ELSE
                   BEGIN
                   %get rid of this page%
                      R1 _ -1;
                      R2.LH _ 4B5;
                      R2.RH _ crpgad[pgindex] / 512;
                      R3 _ 1B6; % get rid of it %
                      !JSYS pmap;
                   ct _ 0;
                   END;
          BUMP &ct;
          END
       UNTIL (pgindex _ pgindex+1) > rfpmax;
       RETURN(hdindex);
       END.

%.....Correspondence list utility procedures....%

    (unlkclist) PROCEDURE; % after unlock file check clist entries
    for validity; clean up if necessary %                          2L1
       LOCAL list, listnd, org;
       POINTER list;
       IF clstad = 0 OR [clstad].clbuff = 0 THEN RETURN;
       list _ [clstad].clbuff;
       listnd _ list + [clstad].clcnt*cll;
       DO
          IF NOT <FILMNP, goodrng>(list.clst1) THEN
             BEGIN
             list.clst2 _ list.clst1;
             list.clst2.stpsid _ origin;
             list.clst1 _ endfil;
             END
       UNTIL (list _ list + cll) = listnd;
       RETURN END.

    (clsid) PROCEDURE %change stid's in passed correspondence list to
    sid's. see clhdr and clistr for format of clist.%              2L2
       (list  % Address of corespondence list %
       );
       %-------------------%
       LOCAL cl, end, fnum;
       REF cl, list;
       end _ (&cl _ list.clbuff) + list.clcnt*cll;
       UNTIL &cl = end DO
          BEGIN
          IF cl.clst1 NOT= endfil THEN
             BEGIN
             fnum _ cl.clst1.stfile;
             cl.clst1.stsid _ getsid(cl.clst1);
             cl.clst1.stsidf _ fnum;
             END;
          IF cl.clst2 NOT= endfil THEN
             BEGIN
             fnum _ cl.clst2.stfile;
             cl.clst2.stsid _ getsid(cl.clst2);
             cl.clst2.stsidf _ fnum;
```

```
        END;
      &ci _ &cl + cll;
      END;
   RETURN;
   END.

(clpsid) PROCEDURE % change sid's in passed correspondence list
to stid's.  Uses lookup to find the ring element with the
corresponding sid. see clhdr and clistr for format of clist.%  2L3
   (list % address of correspondence list %
   );
   %-------------------%
   LOCAL cl, end;
   LOCAL TEXT POINTER ptr;
   REF cl, list;
   ptr _ 0;
   ptr[13 _ 1;
   ptr.stpsid _ origin;
   end _ (&cl _ list.clbuff) + list.clcnt*cll;
   UNTIL &cl = end DO
      BEGIN
      IF cl.clst1 NOT= endfil THEN
         BEGIN
         ptr.stfile _ cl.clst1.stsidf;
         lookup($ptr, cl.clst1.stsid, sid);
         IF (cl.clst1 _ ptr) = endfil THEN
            err($"NLS system error");
         END;
      IF cl.clst2 NOT= endfil THEN
         BEGIN
         ptr.stfile _ cl.clst2.stsidf;
         lookup($ptr, cl.clst2.stsid, sid);
         IF (cl.clst2 _ ptr) = endfil THEN
            err($"NLS system error");
         END;
      &cl _ &cl + cll;
      END;
   RETURN;
   END.

(ciflup) PROCEDURE % change clist references to oldfil to newfil
(assumes list in sid format) %                                  2L4
   (oldfil, % number of file whose references are to be changed %
   newfil,  % NLS file number of new file %
   list     %Address of correspondence list %
   );
   %-------------------%
   LOCAL cl, end;
   REF cl, list;
   end _ (&cl _ list.clbuff) + list.clcnt*cll;
   UNTIL &cl = end DO
      BEGIN
      IF cl.clst1 NOT= endfil THEN
         IF cl.clst1.stsidf = oldfil THEN cl.clst1.stsidf _
         newfil;
      IF cl.clst2 NOT= endfil THEN
```

```
            IF cl.clst2.stsidf = oldfil THEN cl.clst2.stsidf _
            newfil;
         &cl _ &cl + cll;
         END;
      RETURN;
      END.

%.....partial copy and locking utility routines.....%

   (lkun) PROCEDURE (fl, astrng);                                    2M1
      %Resets user setable word to unlock%
      LOCAL lokdir, lokinit;
      REF fl, astrng;
      R1 _ fl.florig;
      R2 _ 1000024B; %read one word at 24B%
      R3 _ $R3;
      !JSYS gtfdb;
      lokdir _ R3.lkdirn;
      lokinit _ R3.lkinit;
      !JSYS gjinf;
      IF R1 = lokdir AND lokinit = cinit THEN
         IF NOT setfdb(0, 0, &fl) THEN
            %If user setable word has not been set back here, file
            is probably bad%
            *astrng* _ *astrng*, "File is bad";
      RETURN;
      END.

   (lockfile) PROCEDURE(fileno);                                     2M2
      IF NOT lkfile(flntadr(fileno)) THEN
         BEGIN
         dismes(2, $lit);
         RETURN(FALSE);
         END;
      RETURN(TRUE)
      END.

   % 'lkfile' moved to NSW affect routines branch %
   (lockid) PROCEDURE(astr, dirnum, initials);                       2M4
      %append "Being Modified By dirnumnam (init)" to astr%
      REF astr;
      LOCAL STRING intstr[5];
      LOCAL cnt, byt;
      *astr* _ *astr*, " Being Modified By ";
      IF dirnum AND SKIP !dirst(byt _ chbptr(astr.L) + &astr,
      dirnum) AND
         (cnt _ slngth(byt, R1)) + astr.L <= astr.M THEN
            astr.L _ astr.L + cnt;
      *intstr* _ NULL;
      IF initials .A 4B6 THEN %old style initials%
         BEGIN
         intstr[1].oldint _ initials;
         intstr.L _ 3;
         END
      ELSE trnsint(initials, $intstr);
      *astr* _ *astr*, SP, "(, *intstr*, ");
```

```
    RETURN END.

(unlktile) PROCEDURE % Unlock file execution procedure %        2M5
    (fileno, % NLS file number of file to be unlocked %
    checklock  % flag to check locking info -- if off and in
    browse mode, don't check locks %
    );
    LOCAL dirno, fl, lkword;
    LOCAL STRING msgstr[100];
    REF fl;
    &fl _ flntadr(fileno);
    !JSYS gjinf; %get current directory%
    dirno _ R1 := fl.florig;
    IF checklock OR NOT fl.flbrws THEN
        BEGIN
        R2 _ 1000024B; %read word at 24B in fdb%
        R3 _ 3; %and stuff it in R3%
        !JSYS gtfdb;
        lkword _ R3; %lock information%
        IF lkword.lkdirn # 0 AND lkword.lkinit # 0 THEN
            BEGIN
            IF lkword.lkdirn # dirno OR lkword.lkinit # cinit THEN
                err($"You do not have this file locked.");
            END
        ELSE err($"This file is not locked");
        END;
    %used to call ckdiracc to check write access, this directory?%
    IF NOT writeout(fileno, $lit)
    OR NOT delpc(fileno, $lit) THEN err($lit);
    IF lit.L > empty THEN
        BEGIN
        dismes(2, $lit);
        END;
    IF NOT rdhdr(fileno, $lit) THEN err($lit);
    IF lit.L > empty THEN
        BEGIN
        dismes(2, $lit);
        END;
    RETURN END.

(lockmes) PROCEDURE % Type out a message indicating that the file
is locked, and by whom %                                        2M6
    (fileno % file number of file whose lock status will be
        printed. %
    );
    LOCAL fl;
    LOCAL STRING tempsr[150];
    REF fl;
    &fl _ flntadr(fileno);
    *tempsr* _ "File";
    !gtfdb( fl.florig, 1000024B, $R3);
    lockid($tempsr, R3.lkdirn, R3.lkinit);
    dismes(4000, $tempsr);
    RETURN;
    END.
```

```
(oplock)  PROCEDURE % Open lock lock procedure.  Used with system
routines (such as Journal) to prevent locking conflict by setting
and resetting  the global lock flag. %                          2M7
   (onoff % If TRUE, try to set flag.  If FALSE, try to reset
      flag. %
   );
   %Waits for open lock lock to be reset, then sets it and
   returns if onoff is true.
      Otherwise, resets lock, and types error message if it was
      not set.
      After waiting 1 minute for lock to be reset, assumes an
      error and overrides  previous lock%
   LOCAL failcount;
   IF onoff THEN
      BEGIN
      failcount _ 0;
      LOOP
         BEGIN
         IF flagut(4, $setfg) THEN EXIT; %Got It%
         IF (failcount _ failcount + 1) > 120 THEN
            BEGIN %flag has been locked for longer than one
            minute%
            dismes(1, $"Open Lock Flag Overidden--Proceed");
            EXIT LOOP;
            END;
         END;
      END
   ELSE
      IF NOT flagut(4, $rstfg) THEN dismes(1,$"Open Lock Lock
      Fail (Flag Reset)--Proceed");
   RETURN;
   END.

(setfdb)  PROCEDURE % Set the lock fields in an FDB.  Return TRUE
if access permitted by maywrt, FALSE if not%                    2M8
   (dirno,% Directory number of person locking file %
   initls, % Initials of person locking file %
   fl      % Address of file status table entry for the file. %
   );
   LOCAL mask, value;
   REF fl;
   IF NOT maywrt(0, &fl) THEN RETURN(FALSE);
   mask _ 0;  mask.lkinit _ mask.lkdirn _ 36M;
   value.lkdirn _ dirno;  value.lkinit _ initls;
   RETURN( chnfdb( fl.florig, $fdbusw, mask, value) );
   END.

% "makepc" moved to NSW affect routines branch %
(dumpc)  PROCEDURE % Clears or sets all of the pc bits in the
file header ring and data block status tables, depending on
value.  Currently, output and update call it after doing all
their edits. %                                                  2M10
   (fileno, % of file whose PC bits are to be set or reset %
   value % Value to be set in PC bits: TRUE or FALSE. (If TRUE,
      then block is to be taken from PC, if FALSE from original
      file. %
```

```
    );
    LOCAL fhd, rn, rngblk, dt, stdb, sdbblk;
    REF rn, dt;
    % get partial copy header address %
       fhd _ filehead[fileno];
    % first do the ring blocks %
       &rn _ fhd - $filhed + $rngst;
       rngblk _ 0;
       DO
          BEGIN
          IF rn.rfexis THEN rn.rfpart _ value;
          &rn _ &rn + 1;
          END
       UNTIL (rngblk _ rngblk + 1) = rngm;
    % now do the sdb blocks %
       stdb _ 0;
       stdb.stfile _ fileno;
       &dt _ fhd - $filhed + $dtbst;
       sdbblk _ 0;
       DO
          BEGIN
          IF dt.rfexis THEN dt.rfpart _ value;
          &dt _ &dt + 1;
          END
       UNTIL (sdbblk _ sdbblk+1) = dtbm;
    RETURN;
    END.

(delpc) PROCEDURE % delete partial copy of file from user
directory if possible; sets lock fields in fdb to zero
(unlocked), set other flags referring to the existence of the
partial copy to zero %                                          2M11
    (fileno, % NLS file number of file whose partial
       copy is to be deleted %
    astrng   % Address of string for error messages %
    );
    %The partial copy is deleted from the user's directory by this
    routine--the header is not reread%
    %----------------%
    LOCAL jfnflg, ckpjfn, dirno, fl;
    REF fl, astrng;
    &fl _ flntadr(fileno);
    *astrng* _ NULL;
    IF fileno NOT IN [1,filcnt] OR
       filcnt NOT IN [1,filmax] OR
       NOT fl.flexis THEN
       BEGIN
       *astrng* _ "NLS system error";
       RETURN(FALSE);
       END;
    IF NOT fl.flbrws THEN
       BEGIN
       IF NOT fl.flpart THEN
          BEGIN
          !JSYS gjinf;
          dirno _ R1 := fl.florig;
```

```
            R2 _ 1000024B; %read one word at 24B%
            R3 _ 3; %put word into R3%
            !JSYS gtfdb;
            END;
         IF fl.flpart OR (R3.lkinit = cinit AND R3.lkdirn = dirno)
         THEN
            IF NOT setfdb(0, 0, &fl) THEN
               BEGIN
               *astrng* _ "No write access";
               RETURN(FALSE);
               END;
         END;
      IF fl.flpart THEN
         BEGIN
         %delete partial copy file%
         R1 _ fl.flpart .V 4B11;
         IF NOT SKIP !JSYS closf THEN
            *astrng* _ "Cannot close partial copy";
         R1 _ fl.flpart;
         IF NOT SKIP !JSYS delf THEN
            *astrng* _ "Cannot delete partial copy";
         END;
      fl.flpart _ 0;
      filepart[fileno] _ FALSE;
      RETURN(TRUE);
      END.

   %cpcnam in NSW affected routines branch%
%.....access utility routines.....%

   (setaccess)  PROCEDURE % Set the NLS system access for the file
   indicated by fileno to type if access type is legal.  Calls err
   if anything wrong. %                                              2N1
      (fileno, % file number whose access is to be set. %
      type % Access type desired. %
      );
      LOCAL jfn, wrtdnm, wrtini;
      IF type > 7 THEN err($"Illegal Access type");
      %First check for lock and access%
      R1 _ jfn _ [flntadr(fileno)].florig;
      R2 _ 1000024B; R3 _ $R3;
      !JSYS gtfdb;
      IF NOT access .A accmask[R3.acctyp] THEN
         err($"Illegal Access");
      IF (R3.lkinit # 0) OR (R3.lkdirn # 0) THEN
         BEGIN %locked..check by whom%
         wrtdnm _ R3.lkdirn; wrtini _ R3.lkinit;
         !JSYS gjinf;
         IF R1 # wrtdnm OR cinit # wrtini THEN
            err($"File Locked");
         END;
      IF NOT maywrt(fileno) THEN
         err($"No write access");
      %Now it is free to change access%
         R1.LH _ 24B; R1.RH _ jfn;
         R2 _ 0;
```

```
      R2.acctyp _ 36M;
      R3.acctyp _ type;
      !JSYS chfdb;
   RETURN END.

(enablaccess) PROCEDURE % Set NLS system access to file or
global mask for user. (Used for special subsystems , e.g.,
Journal.) Set bit in access mask which corresponds to type. Set
mask in file if fileno # 0, Otherwise set global mask. %      2N2
   (fileno, % File number whose access is to be set; if zero,
      set global access mask. %
   type  % NLS access type %
   );
   LOCAL fl; REF fl;
   IF type > 7 THEN err($"Illegal Access Type");
   IF fileno THEN BEGIN &fl _ flntadr(fileno);
      fl.flaccm _ fl.flaccm .V accmask[type];
   END ELSE access _ access .V accmask[type];
   RETURN END.

(disabaccess) PROCEDURE % see enablaccess for comment %      2N3
   (fileno, % File number whose access is to be disabled; if
   zero,
      set global access mask. %
   type  % NLS access type %
   );
   LOCAL fl, mask; REF fl;
   IF type > 7 THEN err($"Illegal Access Type");
   mask _ -(accmask[type] +1); %one complement%
   IF fileno THEN BEGIN &fl _ flntadr(fileno);
      fl.flaccm _ fl.flaccm .A mask;
   END ELSE access _ access .A mask;
   RETURN END.

(maywrt)     PROCEDURE % Uses chkac JSYS to see if user has
access to the file whose file status table entry address or file
number is passed. %                                          2N4
   (fileno, % file number of file to which access is to be
   checked.
      If 0, assumes file status table address has been passed. %
   fl % Address of filst entry for this file; if 0, the address
      will be computer from the fileno. %
   );
   %returns TRUE if may write on the specified file %
   REF fl;
   IF fileno THEN &fl _ flntadr(fileno);
   %get current user settable word%
      !gtfdb( fl.florig, 1000024B, $R3);
   %return value from attempt to change sign bit in USW in FDB%
      RETURN( chnfdb( fl.florig, 24B, 4B11, R3) );
   END.

(chnfdb)     PROCEDURE % Return true if file fdb changed
properly%                                                    2N5
   (jfn,   %jfn for file%
   offset, %offset into fdb for word to be changed%
```

```
      mask,   %mask for which bits to change%
      value   %new value for word (masked by mask)%
      );
      LOCAL savedispatch;
      %save the old illegal instruction psi dispatch address%
         savedispatch _ ilsdsp;
      %set up new dispatch address%
         ilsdsp _ $chndsp;
      %try to change the fdb%
         R1.LH _ offset;   R1.RH _ jfn;
         !chfdb( R1, mask, value);
      %successful, now cleanup and return TRUE%
         ilsdsp _ savedispatch;
         RETURN (TRUE);
      %not successful, debrk and return FALSE%
         (chndsp):                                              2N5K1
            ilsdsp _ savedispatch;
            [levtab] _ $chnext;
            !debrk();
         (chnext):                                              2N5K2
            RETURN( FALSE );
      END.
%.....privacy routines.....%
   (chprvsts) %change file's privacy status%
   PROCEDURE (fileno, newprvsts);                                  201
      %----------%
      LOCAL setting, stid;
      LOCAL TEXT POINTER tp1, tp2;
      %----------%
      CASE newprvsts OF
         = $pspublic: setting _ FALSE;
         = $psprivate: setting _ TRUE;
         = FALSE: RETURN;
         ENDCASE err (0);
      %force creation of partial copy%
         stid _ 0;
         stid.stpsid _ origin;
         stid.stfile _ fileno;
         FIND SF(stid) ^tp1 CH ^tp2;
         ST tp1 tp2 _ tp1 tp2;
      [filehead [fileno] + $filhde-$filhed].prvsts _ setting;
      RETURN;
      END.

   (rdprvsts) %read file's privacy status%
   PROCEDURE (fileno);                                             202
      %----------%
      LOCAL curprvsts;
      %----------%
      CASE [filehead [fileno] + $filhde-$filhed].prvsts OF
         = FALSE: curprvsts _ $pspublic;
         = TRUE: curprvsts _ $psprivate;
         ENDCASE;
      RETURN (curprvsts);
      END.
```

```
(vrprvacc) %verify ident's privacy-status access to file%
PROCEDURE (fileno, %or% filename, %or% catstid, ident);          203
   %----------%
   LOCAL stid, filestid, outcome, fl, derivedfileno;
   LOCAL STRING docnumber [9];
   LOCAL TEXT POINTER tp1, tp2;
   REF filename, ident, fl;
   %----------%
   %access check disabled, enabled wheel, or system?%
      IF skpachk OR nwheelf .A 4B5 OR libflg THEN RETURN (TRUE);
   %initialize%
      stid _ derivedfileno _ 0;
      INVOKE(privcatch, vrpexit);
   %filename supplied%
      derivedfileno _ (IF &filename THEN
         open (0, &filename) ELSE fileno);
   %fileno supplied%
      outcome _ TRUE;
      IF NOT (stid _ catstid) THEN
         BEGIN
         %public file?%
            IF rdprvsts (derivedfileno) = $pspublic THEN
               GOTO vrpexit;
         %does origin statement give him access?%
            %build stid for origin statement%
               filestid.stpsid _ origin;
               filestid.stfile _ derivedfileno;
            IF NOT getfacc (filestid, 0, $tp1, $tp2)
            OR ckilmem (&ident, $tp1, $tp2, 0) THEN
               GOTO vrpexit;
         %does catalog entry give him access?%
            &fl _ flntadr (derivedfileno);
            IF NOT cnvfndocnum (fl.flastr, $docnumber)
            OR NOT (stid _ gtcatent ($docnumber, 0, 0, FALSE))
            THEN
               BEGIN
               outcome _ FALSE;
               GOTO vrpexit;
               END;
         END;
   %open catalog file supplied%
      outcome _
         IF getcacc (stid, 0, $tp1, $tp2) THEN
            ckilmem (&ident, $tp1, $tp2, 0)
         ELSE TRUE;
   %terminate%
      (vrpexit): IF stid.stfile AND NOT catstid THEN          203L1
         close (stid.stfile := 0);
      IF derivedfileno AND NOT fileno THEN
         close (derivedfileno := 0);
   RETURN (outcome);
   (privcatch) CATCHPHRASE();                                  203N
      BEGIN
      CASE SIGNALTYPE OF
         = aborttype :
            TERMINATE;  % goes to vrpexit %
```

```
            ENDCASE;
         CONTINUE;
         END;
      END.

   (getfacc) %fetch access field from file origin statement%
   PROCEDURE (originstid, fldstr, ptr1, ptr2);                    204
      %----------%
      RETURN (getcacc (originstid, fldstr, ptr1, ptr2));
      END.

   (getcacc) %fetch access field from catalog entry%
   PROCEDURE (entrystid, fldstr, ptr1, ptr2);                     205
      %----------%
      LOCAL retval;
      LOCAL TEXT POINTER z1, z2, z3;
      %----------%
      retval _ FALSE;
      makeptr (entrystid, $z3);
      IF NOT fndunqtxt ($"AccessList:", $z3, $z1, $z2)
      OR NOT (FIND z2 > $NP ^z1 [";] < CH $NP ^z2) THEN
         getcend (entrystid, $z1, $z2)
      ELSE retval _ TRUE;
      stptset (fldstr, ptr1, ptr2, $z1, $z2);
      RETURN (retval);
      END.

   (setfacc) %set access field in file origin statement%
   PROCEDURE (originstid, repstr, ptr1, ptr2);                    206
      %----------%
      RETURN (setcacc (originstid, repstr, ptr1, ptr2));
      END.

   (setcacc) %set access field in catalog entry%
   PROCEDURE (entrystid, repstr, ptr1, ptr2);                     207
      %----------%
      LOCAL repcnt;
      LOCAL TEXT POINTER z1, z2;
      REF repstr, ptr1, ptr2;
      %----------%
      IF NOT &ptr1 THEN getcacc (entrystid, 0, $z1, $z2)
      ELSE FIND ptr1 ^z1 ptr2 ^z2;
      IF *repstr* = *nullfield* THEN %delete the field%
         IF (FIND z1 < [$PT] $NP > ^z1 ["AccessList:"] z2 [";] ^z2)
         THEN
            ST entrystid _ SF(entrystid) z1, z2 SE(entrystid)
         ELSE NULL
      ELSE
         BEGIN
         repcnt _ repstr.L;
         IF *repstr* [repstr.L] = "; THEN PUMP DOWN repcnt;
         IF NOT (FIND z1 < [$PT] $NP > ["AccessList:"]) THEN
            ST entrystid _ SF(entrystid) z1, " AccessList: ",
            *repstr* [1 TO repcnt], ";, z2 SE(entrystid)
         ELSE
            ST entrystid _ SF(entrystid) z1, *repstr* [1 TO repcnt],
```

```
            z2 SE(entrystid);
         END;
      RETURN;
      END.

%.....miscellaneous utility routines.....%

   (writeout) PROCEDURE % write out all pages of a file and PC; all
   pages associated with the file are removed from the user's
   map--written back onto the file.  Returns TRUE is successful,
   FALSE if error with message in string whose address is passed.
   Similar to clrcor.  collect common code. %                    2P1
      (fileno,  % of file to be written out %
      astrng  % Address of string for error message %
      );
      %----------------%
      LOCAL pgindex, ct, fl;
      REF ct, fl, astrng;
      IF fileno NOT IN [1,filcnt] OR filcnt NOT IN [1,filmax]
      THEN
         BEGIN
         *astrng* _ "NLS system error";
         RETURN(FALSE);
         END;
      pgindex _ 1;
      &ct _ $corpst + 1;
      DO
         BEGIN
         IF ct.ctfull AND ct.ctfile = fileno THEN
            BEGIN
            IF ct.ctpnum = 0 AND ct.ctfroz > 1
            OR ct.ctpnum # 0 AND ct.ctfroz > 0 THEN
               BEGIN
               *astrng* _ "NLS system error";
               RETURN(FALSE);
               END;
            %JSYS to get rid of the page%
            R1 _ -1;
            R2.LH _ 485;
            R2.RH _ crpgad[pgindex] / 512;
            R3 _ 0;
            !JSYS pmap;
            ct _ 0;
            END;
         BUMP &ct;
         END
      UNTIL (pgindex _ pgindex+1) > rfpmax;
      &fl _ fintadr(fileno);
      fl.flhead _ filehead[fileno] _ 0;
      RETURN(TRUE);
      END.

   (rdhdr)    PROCEDURE  % read file header %                     2P2
      (fileno, % NLS file number of file whose header is to be
         read into core %
      astrng   % Address of string to be used for error message. %
```

```
);
% To read the header block (block zero) of a file into core,
call
this routine with the file number.  If a partial copy
exists for this file, the header is read from it,
otherwise the header is read from the original file.
The file header block is frozen into core and its address
stored into the filst entry for this file.  If the header
is badand it is from a PC, the file is unlocked and the header
read from the original file.  If there was no PC or if the PC
header was bad and the original file header is bad, the file
is initialized by intfil.  A message is returned in the passed
astring, if something went wrong.  Otherwise the string is set
to null.  Returns TRUE if some header read in (even from an
initialized file), FALSE if system error.  Could also call err
with bfile ("bad file") error number is  the jfn in the file
status table for this file is 0. %
%------------------%
LOCAL errnum,  fl, pgindx, vwd, header, blk, db, rn, end, jfn;
LOCAL tmp;
REF astrng, fl, vwd, db, rn;
&fl _ flntadr(fileno);
IF &astrng = 0 THEN &astrng _ $lit;
*astrng* _ NULL;
IF fileno NOT IN [1,filcnt] OR
   filcnt NOT IN [1,filmax] OR
   NOT fl.flexis THEN
   BEGIN
   *astrng* _ "NLS system error";
   RETURN(FALSE);
   END;
% get a page %
   pgindx _ lodrfb (0, -1);
   corpst[pgindx].ctfile _ fileno;
   frzblk (pgindx, 1);
   INVOKE (sigrlse);
% PMAP the header %
   IF jfn _ fl.flpart THEN %get it from the PC%
      R3 _ ( IF NOT fl.flpcread THEN 14B10 %rw access%  ELSE
      100488 %read access only% )
   ELSE
      BEGIN %get it from the original file%
      IF NOT jfn _ fl.florig THEN err(bfile);
      R3 _ 100488; %private copy on write%
      END;
   blk _ crpgad[pgindx];
   R1.LH _ jfn;
   R1.RH _ 0;
   !rpacs(); %see if page exists%
   IF R3 .A 4B10 OR R2 .A 1B10 THEN
      BEGIN
      R2.RH _ blk / 512;
      R2.LH _ 4B5;
      % R3 already set up %
      !JSYS pmap;
      END;
```

```
        fl.flhead _ pgindx;
        &vwd _ (header _ filhdr(fileno)) - $filhed + $nlsvwd;
        fileheadrfileno] _ header;
        errnum _ 0;
        IF [blk].fbtype NOT= hdtyp OR
        [blk].fbpnum NOT= 0 OR
        vwd # nlsvwd THEN
           IF fl.flpart THEN
              BEGIN
              errnum _ 1;
              IF NOT writeout(fileno, &astrng)
              OR NOT delpc(fileno, &astrng) THEN
                 RETURN(FALSE);
              DISABLE (sigrlse);
              %read header from original file%
              %get a new page%
              pgindx _ lodrfb(0, -1);
              corpst[pgindx].ctfile _ fileno;
              frzblk(pgindx, 1);
              ENABLE (sigrlse);
              IF NOT R1.LH _ fl.florig THEN
                 BEGIN
                 *astrng* _ "NLS System Error";
                 RETURN(FALSE);
                 END;
              R1.RH _ 0;
              R3 _ 1004B8; %private copy on write%
              blk _ crpgad[pgindx];
              R2.RH _ blk/512;
              R2.LH _ 485;
              !JSYS pmap;
              fl.flhead _ pgindx;
              &vwd _ (header _ filhdr(fileno)) - $filhed + $nlsvwd;
              fileheadrfileno] _ header;
              IF [blk].fbtype # hdtyp OR
              [blk].fbpnum # 0 OR
              vwd # nlsvwd THEN
                 BEGIN
                 errnum _ 2;
                 intfil(fileno);
                 END;
              END
           ELSE
              BEGIN
              !gtfdb( fl.florig, 186+$fdbbyv, $tmp);
              IF tmp.RH = 0 OR (tmp.RH = 1 AND iszeropage(blk)) THEN
                 BEGIN
                 errnum _ 2;
                 intfil(fileno);
                 END
              ELSE
                 BEGIN
                 *astrng* _ *[fl.flastr]*, " is not an NLS file";
                 RETURN( FALSE );
                 END;
              END;
```

```
    [header + $rfbs + 1 -$filhed] _ 0; %ca bits%
    end _ (&rn _ $rngst - $filhed + header) + rngm;
    DO
       BEGIN
       rn.rfcore _ FALSE;
       IF NOT fi.flpart THEN rn.rfpart _ FALSE;
       END
    UNTIL (&rn _ &rn + 1) >= end;
    end _ (&db _ $dtbst - $filhed + header) + dtbm;
    DO
       BEGIN
       db.rfcore _ FALSE;
       IF NOT fi.flpart THEN db.rfpart _ FALSE;
       END
    UNTIL (&db _ &db + 1) >= end;
    CASE errnum OF
       = 1: *astrng* _ "PC bad, file unlocked";
       = 2: *astrng* _ NULL; %used to be file initialized%
       ENDCASE;
    DROP (sigrlse);
    RETURN(TRUE);

    (sigrlse) CATCHPHRASE;                                          2P2AL
       BEGIN
       DISABLE (sigrlse);
       CASE SIGNALTYPE OF
          =aborttype:
             frzblk(pgindx, -1);
       ENDCASE;
       CONTINUE;
       END;

    END.

(iszeropage) PROCEDURE (page);  %return true if page pointed to
by PAGE is all zeroes, FALSE if not.  note: dos not check for
valid page boundary.%                                               2P3
    REF page;
    LOCAL i;
    FOR i _ 0 UP UNTIL > 511 DO
       IF page[i] THEN RETURN(FALSE);
    RETURN(TRUE);
    END.

(delovsrns) PROCEDURE % Deletes old versions of file if there are
more than the number passed as the second parameter %               2P4
       (jfn,  % of file whose old versions are to be deleted %
       number % of versions to keep %
       );
       R1 _ jfn;
       R2 _ number;
       IF NOT SKIP !JSYS delnf THEN
          BEGIN
          dismes(2, $"unable to delete old versions");
          END;
       RETURN END.
```

```
(dltfile)   % CL: ;  delete a file given the jfn %
PROCEDURE (jfn);                                                    2P5
   % Procedure description
      FUNCTION
         delete a file given the jfn, but don't release the jfn.
      ARGUMENTS
         jfn--INTEGER-jfn of file to be deleted.
      RESULTS
         proc-value--always TRUE.
      NON-STANDARD CONTROL
         calls err if JSYS fails.
      %
   % Declarations %
   % invoke delf JSYS %
      IF NOT SKIP !delf(jfn .V 4B11) THEN err($"Cannot delete
      file");
   % Return %
      RETURN;
   END.

(getversn) PROCEDURE % returns the version number associated with
the file name passed it %                                           2P6
   (string % Address of string containing file name %
   );
   REF string;
   CCPOS SF(*string*);
   *vrsnno* _ NULL;
   IF FIND C".] C(";/".)] ^p1 THEN
      *vrsnno* _ p1 SE(p1);
   IF vrsnno.L = empty THEN RETURN(FALSE);
   RETURN( cvsno($vrsnno) );
   END.

(jfnstr)   PROCEDURE % puts full file name for jfn passed into
string %                                                            2P7
   (jfn, % of file whose name is to be placed into a string %
   string % Address of string into which name will be placed. %
   );
   REF string;
   RETURN (jfntostr (jfn, &string, jfnstf) );
   END.

(jfntostr) PROCEDURE % converts jfn to a full file name %           2P8
   (jfn,     % the jfn %
   string,   % string into which to put the file name %
   format);  % format of file name to be put out by jfns JSYS %
   LOCAL bytptr, count, temp;
   LOCAL STRING !str[100];
   REF string;
   R1 _ bytptr _ chbptr(empty) + $!str;
   R2 _ jfn;
   R3 _ format;
   !JSYS jfns;
   temp _ R1;
   count _ slngth (bytptr, temp);
```

```
    IF string.M < count THEN err ($"NLS system error");
    istr.L _ count;
    *string* _ *istr*;
    RETURN;
    END.

% 'jfntlink' moved to NSW affected routines branch %
(incrvrsn) PROCEDURE (string);                                  2P10
    %Given the addreaa of a string containing a file name, this
    routine will increment the number in the string representing
    the version  number  (if the striing has no ";nn" in it, the
    routine returns FALSE.%
    %-------------%
    LOCAL TEXT POINTER tptr;
    LOCAL number, jfn, flgbits;
    REF string;
    CCPOS SF(*string*);
    IF FIND [".] [(";/".)] ^tptr THEN
       BEGIN
       *string* _ SF(*string*) tptr;
       %get new file name%
          flgbits _ getgtjflg(write, origff, dfltvrs);
          IF NOT jfn _ sgtjfn(flgbits, &string, $lit) THEN
             BEGIN
             *string* _ "******";
             RETURN(FALSE);
             END;
          jfnstr(jfn, &string);
          reljfn (jfn);
       RETURN(TRUE);
       END
    ELSE
       BEGIN
       *string* _ "******";
       RETURN(FALSE);
       END
    END.

(gdname)   PROCEDURE % Convert directory number and into a
directory string.  Calls err if error occurs. %                 2P11
    (dirno, % TENEX directory number %
    dname   % Address of string into which directory name is to
       be placed %
    );
    LOCAL length, bp;
    REF dname;
    bp _ R1 _ chbptr(empty) + &dname;
    R2 _ dirno;
    IF NOT SKIP !JSYS dirst THEN err($"System error");
    length _ slngth(bp, R1);
    IF length > dname.M THEN err($"NLS System error");
    dname.L _ length;
    RETURN END.

(gdftdir)  PROCEDURE % Given a filenumber, puts owner name in a
string %                                                        2P12
```

```
    (fileno, % File number whose owner name is sought %
    string  % Address of string into which owner name is to be
       placed %
    );
    %changed to handle a pointer in funo (with sign bit) to string
    in file header block instead of file number in funo. 7-DEC-73
    23:09 JDH%
    %returns FALSE if file number is not valid -- used for FTP'd
    files%
    LOCAL unum, bytptr, unstr, filhaddr;
    REF string, unstr;
    unum _ [($funo - $filhed) + (filhaddr _ filhdr(fileno))];
    IF unum < 0
       THEN BEGIN
          filhaddr _ filhaddr .A 777000B;
          &unstr _ filhaddr + unum.RH;
          *string* _ *unstr*;
          END
       ELSE BEGIN
          R1 _ bytptr _ chbptr(empty) + &string;
          R2 _ unum;
          IF NOT SKIP !JSYS dirst THEN
             RETURN(FALSE);;
          IF (string.L _ slngth(bytptr, R1)) > string.M THEN
             err($"User name too long.");
          END;
    RETURN;
    END.

(checkjfile) PROCEDURE %convert filename to absolute form via
catalogs%                                                    2P13
    (filename, flags, thorough);
    %----------%
    LOCAL tempjfn, catstid;
    LOCAL STRING docnumber [9], tempfn [40];
    REF filename;
    %----------%
    %skip catalog search unless read access requested%
       IF NOT flags.LH .A gtjred THEN RETURN (FALSE);
    %if superficial search, skip catalogs if directory specified%
       IF NOT thorough AND *filename* [1] = "< THEN RETURN
       (FALSE);
    %syntactically a journal document?%
       IF NOT cnvfndocnum (&filename, $docnumber) THEN
          RETURN (NOT thorough);
    %initialize%
       catstid _ 0;
       INVOKE (sigcls);
    %search catalogs for document%
       IF NOT (catstid _ gtcatent ($docnumber, 0, 0, FALSE)) THEN
          BEGIN %search <TEJOURNAL>%
          *tempfn* _ "<TEJOURNAL>J", *docnumber*, ".NLS", EOL;
          IF NOT (tempjfn _ sgtjfn (flags, $tempfn, 0)) THEN
             BEGIN
             DROP (sigcls);
             RETURN (NOT thorough);
```

```
            END;
         reljfn (tempjfn);
         IF NOT vrprvacc (0, $tempfn, 0, $initsr) THEN
            err ($"Private document; access denied to you");
         *filename* _ *tempfn*;
         DROP (sigcls);
         RETURN (TRUE);
         END;
   %verify access to document%
      IF NOT vrprvacc (0, 0, catstid, $initsr) THEN
         err ($"Private document; access denied to you");
      acchecked _ TRUE;
   %fetch document's filename%
      IF NOT getcfn (catstid, &filename, 0, 0) THEN
         err ($"Hardcopy document; not available on-line");
      IF *filename* [filename.L] # EOL THEN
         *filename* _ *filename*, EOL;
   %close catalog file%
      close (catstid.stfile := 0);
   %terminate%
      dismes (2, $"Catalog File");
      DROP (sigcls);
      RETURN (TRUE);

   (sigcls) CATCHPHRASE;                                          2P13Q
      BEGIN
      DISABLE (sigcls);
      CASE SIGNALTYPE OF
         =aborttype:
            IF catstid.stfile THEN
               sigclose (catstid.stfile := 0);
      ENDCASE;
      CONTINUE;
      END;

   END.

(cnvfndocnum) %convert filename to document number%
PROCEDURE (filename, docnumber);                                  2P14
   %----------%
   LOCAL TEXT POINTER tp1, tp2;
   REF filename, docnumber;
   %----------%
   IF NOT FIND SF(*filename*) ('< ["'>] / ) ^tp1 4$D ^tp2 ('.
   ("NLS" (EOL/ENDCHR) / ENDCHR / EOL) / EOL / ENDCHR)
      THEN RETURN (FALSE);
   *docnumber* _ tp1 tp2;
   RETURN (TRUE);
   END.

(filsize) % find file size %
PROCEDURE ( jfn );                                                2P15
   %
      FUNCTION:
         compute the file's size in pages
      ARGUMENTS:
```

```
            ljfn - integer - file handle
         RESULTS
            res1 - file size in pages or FALSE if file doesn't exist

            res2 - file size in bytes.
      %
      IF SKIP !sizef( ljfn ) THEN RETURN( R3, R2) ELSE RETURN( FALSE
      );
      END.
   (relfsize) % find number of words a rel file will occupy when
   loaded %
   PROCEDURE ( jfn );                                              2P16
      % last byte in rel file is set by compiler to be number of
      words %
      LOCAL lastbyte, size;
      filsize( jfn : lastbyte ); % lastbyte = number of file bytes
      -1 %
      BUMP DOWN lastbyte;
      !rin( jfn, R2, lastbyte);
      size _ R2;
      !rin( jfn, R2, 0); % reset file pointer %
      RETURN ( size);
      END.

%.....FTP routines.....%
   REF ftperm, ftpoh;
   %remote load public file routine%
      (fetchfile) %get a public remote file%                       2Q2A
         PROCEDURE
            %FORMALS%
               (hostno,       %host number astring%
               locfile,       %new name for local file%
               remfile);      %name of file to be loaded%
            LOCAL STRING msg[100], filestr[200], locflnm[200];
            LOCAL sysmsg;
            REF remfile, locfile;
         ftpbgn();
         INVOKE (sigftp, badacctok);
         ftpopn(hostno);
         dismes(1,$"connected");
         ftplog($"ANONYMOUS", $initsr, $"3");
         (badacctok): NULL;                                        2Q2A7
         *msg* _ "Transfering ", *remfile*, " (<CTRL-G> aborts)";
         dismes(1, $msg);
         ftprtr(&locfile, &remfile, $"COMPRESSED");
         ftpcls();
         ftpend();
         DROP (sigftp);
         RETURN;

         (sigftp) CATCHPHRASE;                                     2Q2A16
            BEGIN
            DISABLE (sigftp);
            CASE SIGNALTYPE OF
               = aborttype:
                  BEGIN
```

```
                INVOKE (sigmes, badacctok);
                ftpend ();
                DISABLE (sigmes);
                END;
            ENDCASE;
            END;

        (sigmes) CATCHPHRASE(:sysmsg);                          2Q2A18
            BEGIN
            DISABLE (sigmes);
            CASE SIGNALTYPE OF
               = aborttype:
                BEGIN
                IF %sysgnl = $ftpsig AND% FIND SF(*[sysmsg]*)
                ["Account"] THEN TERMINATE;
                % dismes(1,$"cannot load remote file"); this
                causes a core dump %
                IF (sysmsg := 0) THEN dismes(1,sysmsg);
                END;
            ENDCASE;
            CONTINUE;
            END;

        END.

    %FTP service routines%
        (ftpbgn) PROCEDURE;                                     2Q3A
            %initialize for FTP activity%
            %----------%
            ftpem _ $emsignal;
            INVOKE (sigftp);
            &ftperm _ getstring (100, $dspblk);
            &ftpoh _ getstring (35, $dspblk);
            ftpcrt ();
            ftpex ($"BGN", 0);
            ftpoha _ ftpoh.L _ 0;
            ftpsem ($emreset);
            DROP (sigftp);
            RETURN (TRUE);

            (sigftp) CATCHPHRASE;                               2Q3A14
                BEGIN
                DISABLE (sigftp);
                CASE SIGNALTYPE OF
                   = aborttype:
                      ftpend();
                ENDCASE;
                CONTINUE;
                END;

            END.

        (ftpend) PROCEDURE;                                     2Q3B
            %end of FTP activity%
            %----------%
            ftpem _ $emrtnfalse;
```

```
    IF ftphnd THEN
       BEGIN
       ftpex ($"END", 0);
       ftpkil ();
       END;
    IF &ftperm THEN freestring ((&ftperm := 0), $dspblk);
    IF &ftpoh THEN freestring ((&ftpoh := 0), $dspblk);
    RETURN (TRUE);
    END.

(ftpopn) PROCEDURE (host);                                          2Q3C
    %open distant file system%
    %----------%
    LOCAL outcome;
    REF host;
    %----------%
    IF (outcome _ ftpex ($"OPEN", 1, &host)) THEN
       BEGIN
       ftpoha _ 1; %temp till FTPFRK fixed%
       *ftpoh* _ *host*; %temp till FTPFRK fixed%
       END;
    RETURN (outcome);
    END.

(ftpcls) PROCEDURE;                                                 2Q3D
    %close distant file system%
    %----------%
    LOCAL outcome;
    %----------%
    IF (outcome _ ftpex ($"CLOS", 0)) THEN ftpoha _ ftpoh.L _
    0;
    RETURN (outcome);
    END.

(ftplog) PROCEDURE (user, pass, acct);                              2Q3E
    %login at distant system%
    %----------%
    RETURN (ftpex ($"LOG", 3, user, pass, acct));
    END.

(ftpdml) PROCEDURE (locfile, user);                                 2Q3F
    %send mail to distant user%
    %----------%
    RETURN (ftpex ($"DML", 2, user, locfile));
    END.

(ftplml) PROCEDURE (locfile, user);                                 2Q3G
    %send mail to local user%
    %----------%
    RETURN (ftpex ($"LML", 2, user, locfile));
    END.

(ftpqml) PROCEDURE (locfile, user, host, stagingdir);               2Q3H
    %queue mail for user%
    %----------%
    RETURN (ftpex ($"QML", 4, locfile, stagingdir, host,
```

```
    user));
    END.

(ftpfml) PROCEDURE (locfile, author, authorhost, title,
header, msg);                                                    203I
    %format mail%
    %----------%
    RETURN (ftpex ($"FML", 6, locfile, author, authorhost,
    title, header, msg));
    END.

(ftpsdd) PROCEDURE (directory);                                  203J
    %set distant default directory%
    %----------%
    RETURN (ftpex ($"SDD", 1, directory));
    END.

(ftpelim) PROCEDURE (locfile);                                   203K
    %eliminate local file%
    %----------%
    RETURN (ftpex ($"ELIM", 1, locfile));
    END.

(ftpvhst) PROCEDURE (host);                                      203L
    %verify host%
    %----------%
    RETURN (ftpex ($"VHST", 1, host));
    END.

(ftpsto) PROCEDURE (locfile, disfile, xfermode);                 203M
    %send file to distant system%
    %----------%
    RETURN (ftpex ($"STOR", 3, disfile, locfile, xfermode));
    END.

(ftprtr) PROCEDURE (locfile, disfile, xfermode);                 203N
    %retrieve file from distant system%
    %----------%
    RETURN (ftpex ($"RETR", 3, disfile, locfile, xfermode));
    END.

(ftpapp) PROCEDURE (locfile, disfile, xfermode);                 203O
    %append to file at distant system%
    %----------%
    RETURN (ftpex ($"APP", 3, disfile, locfile, xfermode));
    END.

(ftpdir) PROCEDURE (locfile, remdsg);                            203P
    %retrieve distant directory listing%
    %----------%
    RETURN (ftpex ($"DIR", 2, remdsg, locfile));
    END.

(ftpdel) PROCEDURE (disfile);                                    203Q
    %delete distant file%
    %----------%
```

```
      RETURN (ftpex ($"DEL", 1, disfile));
      END.

   (ftpren) PROCEDURE (curfile, newfile);                          2Q3R
      %rename distant file%
      %----------%
      RETURN (ftpex ($"REN", 2, curfile, newfile));
      END.

   (ftpsem) PROCEDURE (mode);                                      2Q3S
      %set FTP-primitive error mode%
      %----------%
      INVOKE (sigrtn, rtnf4);
      CASE mode OF
         = $emrtnfalse, = $emsignal: ftpem _ mode;
         = $emreset: ftpem _ $emsignal;
         ENDCASE err ();
      DROP (sigrtn);
      RETURN (TRUE);
      (rtnf4):                                                     2Q3S7
      DROP (sigrtn);
      RETURN (FALSE);

      (sigrtn) CATCHPHRASE;                                        2Q3S11
         BEGIN
         DISABLE (sigrtn);
         CASE SIGNALTYPE OF
            = aborttype:
               IF ftpem = $emrtnfalse THEN TERMINATE
               ELSE CONTINUE;
         ENDCASE;
         END;

      END.

   (ftprem) PROCEDURE;                                             2Q3T
      %read FTP-primitive error mode%
      %----------%
      RETURN (ftpem);
      END.

   (ftpoh) PROCEDURE (host);                                       2Q3U
      %read open host name and address%
      %----------%
      REF host;
      IF &host THEN *host* _ *ftpoh*;
      RETURN (ftpoha);
      END.

%FTPFRK driver%
   (ftpex) PROCEDURE (op, count, parm1, parm2, parm3, parm4,
   parm5, parm6);                                                  2Q4A
      %execute FTPFRK operation%
      %----------%
      INVOKE (sigrtn, rtnf5);
      CASE count OF
```

```
            IN [0,2]: ftpcf (op, count, parm1, parm2);
            IN [3,4]:
               BEGIN
               ftpcf ($"ARG", 2, parm1, parm2);
               ftpcf (op, count-2, parm3, parm4);
               END;
            IN [5,6]:
               BEGIN
               ftpcf ($"ARG", 2, parm1, parm2);
               ftpcf ($"ARG", 2, parm3, parm4);
               ftpcf (op, count-4, parm5, parm6);
               END;
            ENDCASE ftperr ($"Invalid FTPFRK argument count.");
         DROP (sigrtn);
         RETURN (TRUE);

         (rtnf5):                                                  2Q4A8
         DROP (sigrtn);
         RETURN(FALSE);

         (sigrtn) CATCHPHRASE;                                     2Q4A12
            BEGIN
            DISABLE (sigrtn);
            CASE SIGNALTYPE OF
               = aborttype:
                  IF ftpem = $emrtnfalse THEN TERMINATE
                  ELSE CONTINUE;
            ENDCASE;
            END;

         END.

   %fork manipulaton subroutines%
      (ftpcrt) PROCEDURE;                                          2Q5A
         %create FTP inferior fork%
         %----------%
         LOCAL jfn;
         LOCAL STRING errstr [100];
         %----------%
         IF ftphnd THEN ftperr ($"FTP already initialized.");
         INVOKE (sigkil);
         IF NOT (jfn _ sgtjfn (getgtjflg (read, 0, oldvrsn),
         $"<NETSYS>FTPFRK.SAV", $errstr)) THEN ftperr ($errstr);
         R1 _ 2B11;
         IF NOT SKIP ! JSYS cfork THEN ftperr ($"Can't create
         inferior fork for FTP.");
         ftphnd _ R1;
         ! HRLS 1;
         ! HRR 1,jfn;
         ! JSYS get;
         DROP (sigkil);
         RETURN;

         (sigkil) CATCHPHRASE;                                    2Q5A18
            BEGIN
            DISABLE (sigkil);
```

```
      CASE SIGNALTYPE OF
         = aborttype:
            ftpkil();
      ENDCASE;
      CONTINUE;
      END;

   END.

(ftpkil) PROCEDURE;                                                    2Q5B
   %kill FTP inferior fork%
   %----------%
   IF ftphnd THEN
      BEGIN
      P1 _ ftphnd;
      ! JSYS kfork;
      ftphnd _ 0;
      END;
   RETURN;
   END.

(ftpcf) PROCEDURE (op, count, parm1, parm2);                           2Q5C
   %invoke FTP inferior fork%
   %----------%
   REF op, parm1, parm2;
   LOCAL STRING errstr [100];
   LOCAL frkacs [16];
   %----------%
   IF NOT ftphnd THEN ftperr ($"FTP not yet initialized.");
   %op code%
      IF op.L > 4
         THEN ftperr ($"FTP op code exceeds four characters.")
      ELSE
         BEGIN
         frkacs _ 0;
         ftppts (&op, $frkacs);
         END;
   %argument count%
      IF count NOT IN [0,2]
         THEN ftperr ($"Invalid number of arguments to
         FTPFRK.")
      ELSE frkacs [1] _ count;
   %parameter 1%
      IF count > 0 THEN
         BEGIN
         IF parm1.L > 34
            THEN ftperr ($"FTP parameter exceeds 34
            characters.");
         ftppts (&parm1, $frkacs [2]);
         END;
   %parameter 2%
      IF count > 1 THEN
         BEGIN
         IF parm2.L > 34
            THEN ftperr ($"FTP parameter exceeds 34
            characters.");
```

```
            ftppts (&parm2, $frkacs [9]);
            END;
        %start fork%
           R1 _ ftphnd;
           R2 _ $frkacs;
           ! JSYS sfacs;
           R2 _ 0;
           ! JSYS sfrkv;
        %wait for completion%
           ! JSYS wfork;
        %check outcome%
           LOOP
              BEGIN
              !rfsts(ftphnd);
              IF (R1 .A 377777B6) THEN EXIT LOOP;
              !disms(100);
              END;
           ! HLRZS 1;
           IF R1 # 2 THEN ftperr ($"FTP inferior fork terminated
           abnormally.");
           R1 _ ftphnd;
           R2 _ $frkacs;
           ! JSYS rfacs;
           IF NOT frkacs [0] THEN RETURN;
           ftpgts ($errstr, $frkacs [1]);
           ftperr ($errstr);
        END.

%miscellaneous subroutines%
   (ftperr) PROCEDURE (msg);                                    2Q6A
      %dispatch FTP error%
      %----------%
      *ftperm* _ *[msg]*;
      ABORT ($ftpsig, &ftperm);
      END.

   (ftpgts) PROCEDURE (string ,addr);                           2Q6B
      %convert ASCIZ to NLS string%
      %----------%
      REF string;
      LOCAL char, bp;
      bp _ chbmty -1 +addr;
      string.L _ 0;
      UNTIL (char _ ^bp) = 0 DO *string* _ *string*, char;
      RETURN
      END.

   (ftppts) PROCEDURE (string, addr);                           2Q6C
      %convert NLS to ASCIZ string%
      %----------%
      REF string;
      IF NOT string.L THEN RETURN;
      R1 _ chbmty -1 + addr;
      R2 _ chbmty + &string;
      R3 _ - string.L;
      ! JSYS sout;
```

```
        ! IDPB 3,1;
        RETURN
        END.

%Routines dealing with Journal files, directories, etc%
    (hiortn) %return to calling fork%
    PROCEDURE (outcome, hdr, body, trlr);                          2R1
        %----------%
        LOCAL STRING msg [500];
        REF hdr, body, trlr;
        %----------%
        msg.L _ 0;
        IF &hdr THEN *msg* _ *msg*, *hdr*;
        IF &body THEN *msg* _ *msg*, *body*;
        IF &trlr THEN *msg* _ *msg*, *trlr*;
        dismes (2, $msg);
        RETURN;
        *msg* _ *msg*, 0;
        R2 _ chbptr (0) + $msg;
        R1 _ outcome;
        !haltf();
        END.
    (jlog)PROC % Type message to Journal Logging file. %
    (number, typest, identsr);                                     2R2
        %
            Number = Number
            typest is a string which describes use of number
            identsr is the ident of author, etc.
        %
        REF number, typest, identsr;
        LOCAL STRING tempsr[200];
        *tempsr* _ *number*, SP, *typest*, SP, *identsr*, SP;
        dtfrmt(-1, $tempsr); %Add the date/time%
        specttyout(0, $tempsr);
        RETURN;
        END.

    (jolock)PROC;                                                  2R3
        %This procedure checks the global flags which tell the current
        status of te Journal:
            Flag 0: (Password jlock): If on Return True, Otherwise
            False.
                This flag is used to prevent new users fromm entering
                the JOurnal, but does not stop persons who are already
                in the process of using it.
            Flag 1: (password jbfil): If on, a file error has been
            noted in one of the JOurnal files.  Discontinnue the
            current Journal Process immediately with a werr.
        %
        IF jdebug THEN RETURN(FALSE);
        IF flagut(1, $tstfg) THEN
            %Some one has found a bad file somewhere%
            werr($"Journal File System Error");
        RETURN(flagut(0, $tstfg));
        END.
```

```
(lockjo)PROC(type);                                                    2R4
   %Type = 1 for bad file lock (flag 1), and 0 for jlock (flag
   0);
   %
   IF type > 1 THEN err($"Illegal Flag Number");
   flagut(type, $setfg);
   specttyout(0, $"Journal Locked");
    IF type = 1 THEN specttyout(oprtty, $"Journal Locked");
   RETURN;
   END.
(jclosfl)PROC;                                                         2R5
   %Close any journal files which may be open%
   IF jworkstid.stfile THEN close(jworkstid.stfile := 0);
   IF jcatstid.stfile THEN close(jcatstid.stfile := 0);
   IF jrnlstid.stfile THEN closeu(jrnlstid.stfile := 0);
   IF diststid.stfile THEN close(diststid.stfile := 0);
   IF numstid.stfile THEN close(numstid.stfile := 0);
   %Set subsystem name back to regular nls%
      R1 _ nlssbn;
      !JSYS setnm;
   RETURN END.

(jflname)PROC(astr);                                                   2R6
   %This procedure accepts the name part of a Journal file name
   as a parameter, and constructs a full name from it.
   The extension is always .NLS, and the user under whom it is
   stored is Journal if it is running as the real system, and
   duvall otherwise (i.e. if jdebug is true)%
   %The file name is built up in the string 'jnamstr', whose
   address is returned%
   LOCAL STRING tempsr[50];
   REF astr;
   *tempsr* _ *astr*;
   astruc($tempsr);
   IF *tempsr* = "IDENTFILE" THEN
      BEGIN
      IF jdebug THEN *jnamstr* _ "<IDENTS>EXP-IDENTS.NLS"
      ELSE *jnamstr* _ *idnfname*;
      END
   ELSE
      BEGIN
      IF jdebug THEN *jnamstr* _ "<DUVALL>"
      ELSE *jnamstr* _ "<JOURNAL>";
      *jnamstr* _ *jnamstr*, *astr*, ".NLS";
      END;
   RETURN($jnamstr);
   END.

(gtcatent) %locate document in catalogs%
PROCEDURE (docnumber, %opt% catfilename, %or% catfileno,
openformodify);                                                        2R7
   %----------%
   LOCAL stid, catalog, fileno, routine, curskpachk, retval;
   LOCAL STRING catstmtname [10];
   LOCAL TEXT POINTER z1;
   REF docnumber, catfilename;
```

```
        %----------%
        IF catfileno THEN %search for statement in open catalog file%
           BEGIN
           %build text pointer to origin statement%
              stid _ 0;
              stid.stpsid _ origin;
              stid.stfile _ catfileno;
              makeptr (stid, $z1);
           *catstmtname* _ 'J, *docnumber*;
           lookup ($z1, $catstmtname, nametyp);
           RETURN (IF z1 # endfil THEN z1 ELSE FALSE);
           END;
        IF &catfilename THEN %open and search catalog file%
           BEGIN
           IF openformodify THEN
              BEGIN
              routine _ $openlock;
              conjdir (TRUE);
              END
           ELSE routine _ $open;
           fileno _ 0;
           curskpachk _ skpachk;
           INVOKE (sigtrap);
           skpachk _ TRUE;
           fileno _ [routine] (0, jflname (&catfilename));
           IF openformodify THEN fileno _ fileno.stfile;
           IF NOT (stid _ gtcatent (&docnumber, 0, fileno,
           openformodify)) THEN
              BEGIN
              close (fileno := 0);
              IF openformodify THEN conjdir (openformodify _ FALSE);
              END;
           skpachk _ curskpachk;
           DROP (sigtrap);
           RETURN (stid);
           END;
        %try all catalog files%
           IF (stid _ gtcatent (&docnumber, $"Tjcat", 0,
           openformodify))
           OR (stid _ gtcatent (&docnumber, $"Jcat", 0,
           openformodify)) THEN
              BEGIN
              DROP (sigtrap);
              RETURN (stid);
              END;
           CASE srval (&docnumber, 10) OF
              < 12000: catalog _ $"Jcat0";
              < 16000: catalog _ $"Jcat1";
              < 20000: catalog _ $"Jcat2";
              < 24000: catalog _ $"Jcat3";
              < 28000: catalog _ $"Jcat4";
              < 32000: catalog _ $"Jcat5";
              < 36000: catalog _ $"Jcat6";
              < 40000: catalog _ $"Jcat7";
              ENDCASE
                 BEGIN
```

```
            DROP (sigtrap);
            RETURN (FALSE);
            END;
      retval _ gtcatent (&docnumber, catalog, 0, openformodify);
      DROP (sigtrap);
      RETURN (retval);
   (sigtrap) CATCHPHRASE;                                          2R7J
      BEGIN
      CASE SIGNALTYPE OF = aborttype:
         BEGIN
         DISABLE (sigtrap);
         skpachk _ curskpachk;
         IF fileno THEN sigclose (fileno := 0);
         IF openformodify THEN conjdir (openformodify _ FALSE);
         END;
      ENDCASE;
      CONTINUE;
      END;
   END.

(getcfn) %fetch filename from catalog entry%
PROCEDURE (entrystid, fldstr, ptr1, ptr2);                         2R8
   %----------%
   LOCAL linkstid, retval;
   LOCAL TEXT POINTER tp1, tp2;
   REF fldstr;
   %----------%
   retval _ FALSE;
   IF NOT &fldstr THEN err ();
   linkstid _ getsub (entrystid);
   IF NOT (FIND SF(linkstid) [Link"] [*(] ^tp1 _tp1 [*)] ^tp2)
   THEN
      getcend (entrystid, $tp1, $tp2)
   ELSE
      BEGIN
      lnbfls ($tp1, 0, &fldstr);
      *fldstr* _ *fldstr*, ".NLS";
      retval _ TRUE;
      END;
   stptset (0, ptr1, ptr2, $tp1, $tp2);
   RETURN (retval);
   END.

(stptset) %set string and/or text-pointers for get routines%
PROCEDURE (string, nptr1, nptr2, optr1, optr2);                    2R9
   REF string, nptr1, nptr2, optr1, optr2;
   IF &string THEN *string* _ optr1 optr2;
   IF &nptr1 THEN
      FIND optr1 ^nptr1;
   IF &nptr2 THEN
      FIND optr2 ^nptr2;
   RETURN;
   END.

(getcend) %fetch end of catalog entry%
PROCEDURE (entrystid, ptr1, ptr2);                                 2R10
```

```
        %----------%
        REF ptr1, ptr2;
        %----------%
        FIND SE(entrystid) ^ptr1 ^ptr2;
        RETURN;
        END.

    (conjdir)PROC(cflag);                                              2R11
        %IF cflag true, connect to Journal Directory, saving info for
        curren one.   Flag false means connect back%
        LOCAL byt;
        IF cflag THEN
            BEGIN
            !JSYS gjinf;
            jdirn _ R2; %conncted directory number%
            R1 _ 1;
            R2 _ chbptr(0) + (IF jdebug THEN $"DUVALL" ELSE
            $"JOURNAL");
            !JSYS stdir;
            GOTO derr;
            GOTO derr;
            R1.LH _ 0;
            R2 _ chbptr(0) + (IF jdebug THEN $"WSD" ELSE $jnlpsw);
            IF NOT SKIP !JSYS cndir THEN
               (derr): err($"Directory Connect Failed");        2R11C11A
            END
        ELSE
            BEGIN
             IF (R1 _ jdirn) = 0 THEN RETURN;
            R2 _ chbptr(0) + $jpassw;
            IF NOT SKIP !JSYS cndir THEN
               BEGIN
               !JSYS gjinf;
               R2 _ 0;
               IF NOT SKIP !JSYS cndir THEN
                  err($"Connect return failed--left in Journal");
               err($"Connect return failed--returned to Login
               Directory");
               END;
            END;
        RETURN
        END.

    (conidir)PROC(cflag);                                              2R12
        %IF cflag true, connect to Identfile Directory, saving info
        for current one.   Flag false means connect back%
        LOCAL byt;
        IF cflag THEN %connect to <identfile>%
            BEGIN
            !gjinf();
            idirn _ R2; %conncted directory number%
            R1 _ 1;
            R2 _ chbmty + $"IDENTS";
            !stdir();
            GOTO derr2;
            GOTO derr2;
```

```
      R1.LH _ 0;
      R2 _ chbmty + $jnlpsw; %password for <identfile>%
      IF NOT SKIP !cndir() THEN
         (derr2): err($"Directory Connect Failed");        2R12C11A
      END
   ELSE
      BEGIN
       IF (R1 _ idirn) = 0 THEN RETURN;
      R2 _ chbmty + $ipassw;
      IF NOT SKIP !cndir() THEN
         BEGIN
         !gjinf();
         R2 _ 0;
         IF NOT SKIP !cndir() THEN
            err($"Connect return failed--left in directory
            Identfile");
         dismes(2, $"Connect return failed--returned to Login
         Directory");
         END;
      END;
   RETURN
   END.

(condir)PROC(destname, destpassw, retname, retpassw);          2R13
   %Connect to destination directory, and return name and
   password of current directory in indicated strings.
      Return true if ok, false if not%
   LOCAL dirno, byt;
   LOCAL STRING cdirnam[20], cpassw[20], odirnam[20], opassw[20];
   REF destname, destpassw, retname, retpassw;
   %First get name and password of current directory%
      *cpassw* _ *jpassw*; %may have been set by connect
      directory command%
   %now get directory name%
      !JSYS gjinf;
      dirno _ R2;
      byt _ R1 _ chbptr(0) + $cdirnam;
      R2 _ dirno;
      IF NOT SKIP !JSYS dirst THEN
         (conderr):                                        2R13F5A
         RETURN(FALSE);
      cdirnam.L _ slngth(byt, R1);
   %Now connect to new directory%
      *odirnam* _ *destname*, 0;
      *opassw* _ *destpassw*, 0;
      R1 _ 1;
      R2 _ chbptr(0) + $odirnam;
      !JSYS stdir;
         GOTO conderr;
         GOTO conderr;
      R1.LH _ 0;
      R2 _ chbptr(0) + $opassw;
      IF NOT SKIP !JSYS cndir THEN GOTO conderr;
      IF &retname THEN *retname* _ *cdirnam*;
      IF &retpassw THEN *retpassw* _ *cpassw*;
   RETURN(1);
```

```
        END.
%NSW affected routines   -- these are for NLS outside NSW, see
(nsw-sources, nswioexec,) for NSW%                                          3
   (makepc)PROC(fl, fileno, undmodflag, astr);                              3A
      REF fl, astr;
      LOCAL jfn, i, flg, delflag;
      %Create a PC for te indicated file, return any errors in astr.
      REturn TRUE if ok, FALSE otherwise%
      delflag _ FALSE;  %init%
      flg _ getgtjflg(write, chkpcf, 0);
      IF (fl.flpart _ R1 _ sgtjfn(flg, fl.flpcst, &astr)) THEN BEGIN
         R2 _ 1000001B;
         R3 _ $R3;
         !JSYS gtfdb;
         IF SKIP !TLNE R3,60000B AND NOT undmodflag THEN
            BEGIN  %exists already, not deleted, and not called to
            undelete mods%
            thwfil(fileno);
            delfil(fileno);
            IF nlmode = fulldisplay THEN alldsp() ELSE tda.dacsp _
            endfil;
            *astr* _ "File Locking Conflict--Please Reload File";
            RETURN (FALSE, TRUE);
            END;
         IF SKIP !TLNN R3,40000B THEN  %deleted%
            BEGIN %must undelete file%
            !HRLI R1,1; !HRLZI R2,40000B; R3 _ 0; !JSYS chfdb;
            delflag _ TRUE;
            END;
         END;
      IF NOT fl.flpart OR
         NOT sysopen(fl.flpart, readwrite, random, &astr) OR
         NOT sysclose(fl.flpart .V 4B11, &astr) THEN
            BEGIN
            IF (R1 _ fl.flpart) THEN
               BEGIN %get rid of jfn%
               IF SKIP !JSYS closf THEN NULL
               ELSE
                  %closf jsys failed (file may already be closed; set
                  up R1 for rljfn%
                  R1 _ fl.flpart;
               IF SKIP !JSYS rljfn THEN NULL;
               END;
            fl.flpart _ 0;
            filepart[fileno] _ FALSE;
            RETURN(FALSE, FALSE);
            END;
      filepart[fileno] _ TRUE;
      IF NOT sysopen(fl.flpart, readwrite, random, &astr) THEN
         BEGIN %Wait fo 1 sec and try it again in case of a timing
         problem% .
         R1 _ 1000;
         !JSYS dismS;
         IF NOT sysopen(fl.flpart, readwrite, random, &astr) THEN
            BEGIN
            IF (R1 _ fl.flpart) THEN %get rid of jfn%
```

```
            BEGIN
            IF SKIP !JSYS closf THEN NULL
            ELSE
               %closf jsys failed (file may already be closed; set
               up R1 for rljfn%
               R1 _ fl.flpart;
            IF SKIP !JSYS rljfn THEN NULL;
            END;
         fl.flpart _ 0;
         filepart[fileno] _ FALSE;
         RETURN(FALSE, FALSE);
         END;
      END;
   IF undmodflag THEN
      BEGIN  %undeleting modifications%
      IF NOT delflag THEN err($"Modifications not deleted.");
      %set the lock word%
         !gtfdb(fl.florig, 1000024B, $i);
         i.lkinit _ cinit;
         i.lkdirn _ fl.fldirno;
         chnfdb(fl.florig, 24B, -1, i);
      %read in the PC header%
         IF NOT rdhdr(fileno, &astr) THEN err(&astr);
      END
   ELSE
      BEGIN  %regular PC creation%
      % change protection of pc to be same as file %
         !gtfdb(fl.florig, 1000004B, $R3); %protecton of original
         file%
         R1.LH _ 15; %offset for protection word)%
         R1.RH _ fl.flpart; %jfn for partial copy%
         !chfdb(R1, 777777B, R3); %change righthalf bits only%
      %map header out to the pc%
         R1.LH _ 4B5;
         R1.RH _ crpgad[fl.flhead] / 512;
         R2.LH _ fl.flpart;
         R2.RH _ 0;
         R3 _ 14B10;
         !JSYS pmap;
         IF tops20flag THEN
            BEGIN
            !EXCH R1,R2;
            !pmap();
            END;
      %DELETE OLD PAGES EXCEPT FOR THE HEADER%
         jfn _ fl.flpart;
         FOR i _ 1 UP UNTIL >= 512 DO
            BEGIN
            R1.LH _ jfn;  R1.RH _ i;
            !rpacs(R1);
            IF R2 .A 1B10 THEN %page exists -- get rid of it%
               BEGIN
               R2.LH _ jfn;  R2.RH _ i;
               !pmap(-1, R2, 0);
               END;
            END;
```

```
      END;
   RETURN(TRUE, FALSE)
   END.

(lkfile) PROCEDURE(fl);                                                 3B
   LOCAL
      dir, lkword, int, cnt, byt, bptr, bptr1, char, mask, value;
   LOCAL STRING intstr[5], dirname[30];
   REF fl;
   !gtfdb( fl.florig, 1000024B, $R3);
   lkword _ R3;
   IF NOT maywrt(0, &fl)
   OR NOT fl.flaccm .A accmask[lkword.acctyp] THEN
      BEGIN
      *lit* _ "No Write Access To ", *[fl.flastr]*;
      RETURN(FALSE);
      END;
   IF lkword.lkdirn NOT= 0 OR
      lkword.lkinit NOT= 0 THEN %file is locked%
      BEGIN
      dir _ lkword.lkdirn;
      int _ lkword.lkinit;
      %See If user has locked it..if so, let it go%
      !JSYS gjinf;
      IF R1 = dir AND cinit = int THEN RETURN(TRUE);
      IF int .A 4B6 THEN %maybe locked with old style initials%
         BEGIN
         *intstr* _ NULL;
         trnsint(cinit, $intstr);
         IF intstr.L = 3 AND intstr[1].oldint = int THEN
            RETURN(TRUE);
         END;
      *lit* _ *[fl.flastr]*;
      lockid($lit,  dir, int); %get lock ident string%
      RETURN(FALSE);
      END;
   %Now get directory from PC name%
      bptr _ chbptr(0) + fl.flpcst;
      bptr1 _ chbptr(0) + $dirname;
      WHILE  ^bptr # "< DO NULL;
      WHILE (char _^bptr) # "> DO ^bptr1 _ char;
      ^bptr1 _ 0;
      R1 _ 0;
      R2 _ chbmty + $dirname;
      !JSYS stdir;
         GOTO lkfilerr;
         GOTO lkfilerr;
      dir _ R1.RH; %Directory Number%
   mask _ 0;
   mask.lkinit _ mask.lkdirn _ 36M;
   value.lkinit _ cinit;
   value.lkdirn _ dir;
   chnfdb(fl.florig, 24B, mask, value);
   RETURN(TRUE);
   (lkfilerr): err($"Illegal PC Name");                                 3B15
   END.
```

```
(cpcnam)  PROCEDURE % converts the file name passed in oldstg to a
file name for pc in newstng %                                          3C
   (oldstg, % Address of original file name string. %
   dirnum   % Directory number of user creating PC; name will
      go into PC name. %
   );
   REF oldstg;
   LOCAL byt, retadr;
   LOCAL STRING dirnam[25], newstg[200];

   % RETURNS %
      % returns the address of a string  (which will contain the pc
      name) which is obtained via getstring from block dspblk %

   CCPOS SF(*oldstg*);
   IF NOT FIND [*<] ^p1 [*>] ^p2 _p2 ^p3 [*.] ^p4 _p4 [(*;/*.)]
   ^p5 _p5
      THEN err($"NLS system error");
   %get directory name%
      R1 _ byt_ chbptr(empty) + $dirnam;
      R2 _ dirnum;
      IF NOT SKIP !JSYS dirst THEN err($"NLS system error");
      dirnam.L _ slngth(byt, R1);
   *newstg* _*<, *dirnam*, *>,  cpcldelim, p1 p2, cpcrdelim,  p3 p4,
   *.;
   *newstg* _ *newstg*, "PC";
   *newstg* _ *newstg*, p5 SE(*oldstg*);
   retadr _ getstring( newstg.L + 1, $dspblk);
   *[retadr]* _ *newstg*;
   RETURN( retadr );
   END.

(jfnflink)   % converts jfn to file link string %                      3D
   PROCEDURE
      (fjfn, % jfn of file %
      jfnname,       % address of string to receive file link %
      flags  % flags for jfns jsys for which fields to put in link %
      );
   LOCAL TEXT POINTER
      tp1, tp2, tp3, tp4;
   LOCAL STRING
      locstr[200]    % work string %
      ;
   REF jfnname;

   % fields from jfntostr (+ dir & file names & ;T and punctuation)
   %
      jfntostr( fjfn, $locstr, flags .V 011B9 .V 4B4 .V 1);
   % now parse this string %
      IF NOT FIND SF(*locstr*) > [*<] ^tp1 [*>] ^tp2 _tp2 ^tp3
      [ENDCHR] ^tp4 THEN err( $"system screwup");
   % now build the file link %
      *jfnname* _ *<, SP, tp1 tp2, *,, SP, tp3 tp4, *,, SP, *>;
   % all done so go home %
      RETURN;
```

```
   END.

(getdno)   % LP: ;  get directory number given file name %
PROCEDURE (namstr REF % => dirno %);                                    3E
   % Procedure description
      FUNCTION
         Search file name for directory name.  Use "transdir" to get
         directory number.
      ARGUMENTS
         namstr:  address of file name string
      RESULTS
         dirno:  directory number
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % Declarations %
      LOCAL TEXT POINTER ptr1, ptr2;
      LOCAL STRING dirs[40];
   % Get directory name %
      CCPOS SF(*namstr*);
      FIND ["<] ^ptr1 [">] < CH ^ptr2;
      *dirs* _ ptr1 ptr2;
   % Return %
      RETURN(transdir($dirs));
   END.

(lgetjfn)  PROCEDURE   % long gtjfn jsys-- set up registers and
execute long gtjfn.  Returns jfn for file whose name is specified in
the parameters if successful, FALSE with error message in string
whose address is passed if not.  If &dirnam is 0, then the file name
is checked to see if it is a Journal item and the default directory
is taken as the one the user is connected to. %                         3F
   (dirnam,  % string containing default directory -- or 0 %
   flnam,    % string containing the name of the file; may be
      changed by checkjfile if this is a journal item %
   extnsn,   % string containing the default extension %
   flags,    % flags for the gtjfn jsys %
   errstr);  % string in which to return error messages %
   LOCAL temp,jfilef;
   LOCAL STRING lfilnam[200], ldirnam[200];   % so don't change
   actual parameters %
   LOCAL TEXT POINTER tp1;
   REF dirnam, flnam, extnsn, errstr;
   *lfilnam* _ *flnam*;
   IF &dirnam = 0 THEN jfilef _ checkjfile ($lfilnam,flags,FALSE)
   ELSE
      BEGIN
      *ldirnam* _ *dirnam*, 0;
      ldirnam.L _ ldirnam.L - 1;
      END;
   (lgetjl):                                                            3F13
   gtjfnt[0] _ flags;
   gtjfnt[1] _ 377777377777B; %no input or output tty%
```

```
    gtjfnt[2] _ 0; %normal device default%
    gtjfnt[3] _ IF &dirnam = 0 THEN 0 ELSE chbptr (empty) + $ldirnam;
       %directory default%
    gtjfnt[4] _ 0; %normal name default%
    gtjfnt[5] _ chbptr(empty) + &extnsn;
    gtjfnt[6] _ 0; %normal protection default%
    gtjfnt[7] _ 0; %normal account default%
    gtjfnt[8] _ 0; %not used%
    *lfilnam* _ *lfilnam*, 0;
    lfilnam.L _ lfilnam.L - 1;
    R2 _ chbptr(empty) + $lfilnam;
    R1 _ $gtjfnt;
    rubabt _ TRUE;
    IF NOT SKIP !JSYS gtjfn THEN
       BEGIN
       temp _ R1;
       rubabt _ FALSE;
       IF NOT jfilef AND (jfilef_checkjfile($lfilnam,flags,TRUE))
       THEN GOTO lgetjl;
       gtjerr (temp, &errstr, flags);
       IF jfilef THEN
          BEGIN
          FIND SE(*lfilnam*) $NP > ^tp1;
          *errstr* _ SF(*lfilnam*) tp1, ": ", *errstr*;
          END;
       RETURN(FALSE);
       END;
    temp _ R1;
    rubabt _ FALSE;
    RETURN (temp.RH, temp.LH);
    END.

(sgtjfn) PROCEDURE % short gtjfn jsys-- set up registers and execute
short gtjfn.  Returns jfn for file whose name is specified in flnam
if successful, FALSE with error message in string whose address is
passed if not.  The file name is checked to see if it is a Journal
item. %                                                          3G
    (flags, % Flags for gtjfn jsys %
    flnam,  % String containing name of file; may be changed by
       checkjfile if name is a journal item %
    errstr  % String for error message %
    );
    LOCAL errcode, jfn, jfilef;
    LOCAL STRING lfilnam[100];
    REF flnam, errstr;
    %short gtjfn%
    *lfilnam* _ *flnam*;
    jfilef_checkjfile ($lfilnam,flags,FALSE);
    (sgtjf1):                                                    3G11
    *lfilnam* _ *lfilnam*, 0;
    lfilnam.L _ lfilnam.L - 1;
    R2 _ chbptr(empty) + $lfilnam;
    R1 _ flags .V sgtjff;
    IF NOT SKIP !JSYS gtjfn THEN
       BEGIN
       errcode _ R1;
```

```
        IF NOT jfilef AND (jfilef_checkjfile($lfilnam,flags,TRUE))
        THEN GOTO sgtjfl;
        IF &errstr THEN
           BEGIN
           gtjerr (errcode, &errstr, flags);
           *errstr* _ *lfilnam*, ": ", *errstr*;
           END;
        RETURN(FALSE, errcode);
        END;
     jfn _ R1;
     RETURN (jfn.RH, jfn.LH);
     END.

(setdfdir)   % CL: ;  set up default directories %
PROCEDURE;                                                          3H
     % Procedure description
        FUNCTION
           Set up subsystem, output processor and compiler default
           directories.  For NLS outside the NSW all directories but
           the subsystem directory are initialized as constant string.
            The subsystem default directory is set to be that from the
           BE came.
        ARGUMENTS
           none
        RESULTS
           proc-value
        NON-STANDARD CONTROL
           none
        GLOBALS
           subdfdir
        %
     % Declarations %
     % initialize subsystem default directory to be that from which we
     came %
        dirfrom($subdfdir);
     % Return %
        RETURN;
     END.

FINISH of ioexec
   (ckdiracc)   PROCEDURE % given a file number, this routine checks
   that the current job has write access to the directory that the file
   lives in; if not, this routine goes to err with a nasty message %  4A
      (fileno % of file to which access is being sought %
      );
      LOCAL cnt, byt, fl;
      LOCAL STRING msgstr[100];
      REF fl;
      IF tenex >= 13200 THEN RETURN;
      &fl _ flntaddr(fileno);
      !chkac( fl.fldirno );
      IF (R1 .A 208) THEN RETURN;
      *msgstr* _ "No write access to ";
      R1 _ byt _ chbptr(msgstr.L) + $msgstr;
      R2 _ fl.fldirno;
      IF SKIP !JSYS dirst AND
```

```
    (cnt _ slngth(byt, R1)) + msgstr.L <= msgstr.M THEN
      msgstr.L _ msgstr.L + cnt;
  err($msgstr);
  END.
```

(exstc)	<nine, isp, 0234>	PROCEDURE	6E2
(nswsw)	<nine, isp, 0265>	EXT CONSTANT =FALSE	5A
(pcpinit)	<nine, isp, 038>	PROCEDURE	6D
(psirtns)	<nine, isp, 0175>	PROCEDURE	6E1
(termmiddle)	<nine, isp, 026>	PROCEDURE	6C

```
< NINE, ISP.NLS.3, >, 9-Nov-77 09:20 SKO ;;;;
FILE isp % using for nls 8.5 <arcsubsys,x110,> using for nls 9
<arcsubsys,1109,> for <relnine, isp.rel,>%
%Interface code for Shared Page and Raw Network Communication with FE%
ALLOW!
%Compile time switches%
   SET DEBUG = TRUE; %TRUE => code for tracing messages%
% Declarations %
   (nswsw) EXTERNAL CONSTANT = FALSE;  %TRUE if NSW NLS%                5A
   INCLUDE <nine,msg3format,declarations>
% Source-code %
   INCLUDE <nine,msg3format,source-code !common>
   INCLUDE <nine,msg3format,source-code !not-msg3>
   (termmiddle) PROCEDURE;  % Terminate process %                        6C
      % log out job %
         IF disjob THEN  %job created by dispatcher%
            IF NOT SKIP !lgout(-1) THEN !haltf(); %log out failed%
         LOOP !haltf();
         END.
   (pcpinit)   % CL: ;  initializaton for procedure call protocol %
   PROCEDURE ;                                                          6D
      % Procedure description
         FUNCTION
            This is the initialization procedure for raw network
            connection or shared page communication.
            For raw network connections (contents of AC1 at startup is
            5)
               Get the jfn's for the receive and send connections.
               Open the receive and send connections.
               Log in the job if the job has not already been logged
               in.  DISJOB is TRUE if the job has not yet been logged
               in.
               The open is done in connect (active) not listen
               (passive) mode.
               Connect mode is used because it guarantees that the
               connection is made to the particular FE that caused this
               BE to be created.  It eliminates the need to check the
               foreign host or socket or to issue an MTOPR to accept
               the connection.  It also causes a time out if the
               connection is not established within a specific amount
               of time.
               In listen mode a check would have to be made on the
               foreign host and socket numbers and the BE would have to
               do an MTOPR to accept the connection.  This would be
               done when there is a state change.  The BE would have to
               do something (jsys's and checks) to cause a time out to
               occur if the connection was not established within a
               reasonable period of time.
               The mode of the open is determined by the filename used
               in the GTJFN -- the extension is FH-FS (FH: foreignh
               host; FS: foreign socket) indicating connect mode.
               One channel is used for a state change on either
               connection.  Another channel is used for an interrupt on
               the BE receive (FE send) connection.
            For shared page (contents of AC1 at startup is 1)
               FE receive channel is in AC2.  Fork handle is in AC3
```

```
        Arm channels for BE receive and command interrupt (^O)
        Send message to FE when BE ready to receive
    ARGUMENTS
      none
    RESULTS
      none
    NON-STANDARD CONTROL
      Go to err if gtjfn fails
      Go to err if login fails
    GLOBALS
      msginputjfn, msgoutputjfn, rcvchannel
    %
  % Declarations %
    REF exresl;
    LOCAL arglst REF;
    LOCAL STRING strng1[50], strng2[5];
  % Set globals for external calls %
    intext();
  % Determine FE type and establish appropriate connection %
    howpcp _ stacs.LH;  %protocol type in  startup AC0%
    CASE howpcp OF
      = sppcp:  %shared page - FE on same computer as BE%
        BEGIN
        % Get send channel and FE fork handle %
          sndchannel _ stacs[1];  %BE send to FE%
          fhandle _ stacs[2];
        % Arm receive message and interrupt channels %
          armpsi (rcvchannel, $rcvpsi, msgpriority);
          armpsi (intchannel, $intpsi, 3  %lowest priority%);
        % CALL FE procedure to let FE know that BE is ready to
        receive and to give FE the BE receive and command
        interrupt channels.  Call with no acknwledgement
        requested from FE. %
          &arglst _ getlst(2);  %allocate list%
          #arglst# _ rcvchannel, intchannel;  %use global%
          extcall(feident, $"beinfo", noack, &arglst);
        END;
      = rawpcp:  %raw network - FE on PDP 11%
        BEGIN
        % Set up foreign host and sockets from startup ac's %
          fhost _ stacs[1];
          fssckt _ stacs[2] + 3;  %socket on which FE will
          send%
          frsckt _ stacs[2] + 2;  %socket on which FE will
          receive%
        % Arm state change and interrupt channels %
          armpsi (stcchannel, $stcpsi, msgpriority);
          armpsi (intchannel, $intpsi, msgpriority);
        % Set up BE receive connection %
          % get jfn %
            *strng1* _ "NET:0.", STRING(fhost, 8), "-",
            STRING(fssckt, 8), ";T", 0;
            IF NOT SKIP !gtjfn(gjfnflag, chbmty + $strng1)
            THEN
              err($"Getjfn failed - pcpinit");
            msginputjfn _ R1.RH;
```

```
            %Open connection%
               IF NOT SKIP !openf(msginputjfn, opnrflag) THEN
                  err($"Error in openning receive connection -
                  pcpinit");
               rcvrdy _ TRUE;  %check for open if state change%
            %Set channel for connection state change%
               setchn(msginputjfn, stcmtopr);
         % Set up BE send connection %
            % get jfn %
               *strng1* _ "NET:0.", STRING(fhost, 8), "-",
               STRING(frsckt, 8), ";T", 0;
               IF NOT SKIP !gtjfn(gjfnflag, chbmty + $strng1)
               THEN
                  err($"Getjfn failed - pcpinit");
               msgoutputjfn _ R1.RH;
            %Open connection%
               IF NOT SKIP !openf(msgoutputjfn, opnsflag) THEN
                  err($"Error in openning send connection -
                  pcpinit");
               sndrdy _ TRUE;  %check for open if state change%
            %Set channel for connection state change%
               setchn(msgoutputjfn, stcmtopr);
         % Initiate interrupt on state change channel in case
         connection open completes before MTOPR to set channel
         for state change is issued %
            !iic(400000B %this fork%, stcbit);
         %  Wait for both listens to complete %
            LOOP !wait();
         % Login the job %
            (logjob):                                          6D4B2H1
            IF disjob THEN
               BEGIN
               % SHOULD GET LOGIN PARAMETERS FROM FE %
               *strng1* _ *logpas*, 0;  %TENEX string%
               *strng2* _ *logacc*, 0;  %TENEX string%
               IF NOT SKIP !login(logdir, chbmty + $strng1,
               chbmty + $strng2) THEN
                  err($"Error in login jsys - pcpinit ");
               END;
         %Set channel for interrupt on receive connection%
            setchn(msginputjfn, intmtopr);
         END;
      ENDCASE
         BEGIN
         erraddr _ $" Invalid protocol type - pcpinit ";
         GOTO errlog;
         END;
   % Return %
      RETURN;
   END.

% MSG3 PSI Routines and their support routines %
   (psirtns) %DO NOT CALL THIS ROUTINE -- PSI ROUTINE%
   PROCEDURE;                                                  6E1
      (stcpsi):  %change of connection state %                 6E1A
         % here for interrupt when there is a change of connection
```

```
      state on the receive/send connection (channel 4) or from
      IIC.  If here from state change other than completion of
      open, set error flag and if both connections had never
      opened DEBRK to errlog.  If both connections are opened,
      DEBRK to logjob.  Otherwise (if one open connection is
      still in wait state or open has not yet been issued for the
      connection) DEBRK to interrupted code %
      %GLOBALS
         connok:  Set to FALSE if error, TRUE if both connections
         opened, otherwise unchanged
         errstate:  Set to TRUE if error, otherwise unchanged
         rcvrdy, sndrdy:  Used
         ropnd,rrfcs,sopnd, srfcs:  Set
      %
      % save ac's %
         psiac1 _ R1;
         R1 _ $psiacs;
         !BLT R1, psiace;
      % check state change %
         IF rcvrdy THEN  %open for receive has been done%
            BEGIN
            ropnd _ exstc(msginputjfn: rrfcs);
            IF NOT (ropnd OR rrfcs) THEN errstate _ TRUE;
            END;
         IF sndrdy THEN  %open for receive has been done%
            BEGIN
            sopnd _ exstc(msgoutputjfn: srfcs);
            IF NOT (sopnd OR srfcs) THEN errstate _ TRUE;
            END;
         IF ropnd AND sopnd THEN  %both opened%
            BEGIN
            connok _ TRUE;
            [ levtab[msgpriority-1] ] _ $logjob
            END
         ELSE IF errstate THEN  %something went wrong%
            BEGIN  %check if connections were ever opened%
            IF connok THEN connok _ FALSE  %they had been%
            ELSE [ levtab[msgpriority-1] ] _ $errlog;
            END;
      % restore ac's %
         R1.LH _ $psiacs;
         R1.RH _ 0;
         !BLT R1, 17B;
         R1 _ psiac1;
      % return to interrupted code or go to lobjob or errlog %
         !debrk();
   (rcvpsi):  %interrupt from FE - msg set up in shared page% 6E1B
      % save ac's %
         psiac1 _ R1;
         R1 _ $psiacs;
         !BLT R1, psiace;
      CASE rcvchecking OF
         = TRUE: %in checking section of getmsg%
            [ levtab[msgpriority-1] ] _ $rcvcomplete; % change pc
            to past wait %
         = FALSE: % just set flag%
```

```
            rcvrdy _ TRUE;
          ENDCASE;
        % restore ac's %
          R1.LH _ $psiacs;
          R1.RH _ 0;
          !BLT R1, 17B;
          R1 _ psiac1;
        !debrk();
    END.
(exstc)   % CL: ;  examine state change %
PROCEDURE (jfn % => opndflag, rfcsflag %);                          6E2
    % Procedure description
      FUNCTION
        Do a GDSTS and determine if the state change is the
        completion of an open connection or the connection is
        still in the process of being opened.  See JSYS manual
        for details.
      ARGUMENTS
        jfn:  jfn for connection
      RESULTS
        opndflag:  TRUE if state is OPND, otherwise FALSE
        rfcsflag:  TRUE if state is RFCS, otherwise FALSE
      NON-STANDARD CONTROL
        none
      GLOBALS
        none
      %
    % Declarations %
      LOCAL state, opndflag _ FALSE, rfcsflag _ FALSE;
    % check state change for RFCS or OPND %
      !gdsts(jfn);
      state _ R2.LH .A 740000B;  %B0-B3 have state change%
      IF state = 300000B  THEN rfcsflag _ TRUE
      ELSE IF state = 340000B THEN opndflag _ TRUE;
    % Return %
      RETURN(opndflag, rfcsflag);
    END.

FINISH
```

```
(catchsiz)      <110, xl10data, 094>            CONSTANT =75          4F
(catsiz)        <110, xl10data, 095>            CONSTANT =10          4D
(gstack)        <110, xl10data, 023>            EXT                   7F
(gstkdesc)      <110, xl10data, 0128>           EXT                   7E
(gstksz)        <110, xl10data, 039>            EXT CONSTANT =4608    4F
(pstksz)        <110, xl10data, 038>            EXT CONSTANT =20      4G
(signalsiz)     <110, xl10data, 092>            CONSTANT =10          4A
(sigsiz)        <110, xl10data, 093>            CONSTANT =12          4C
(stackdecsize)  <110, xl10data, 0153>           CONSTANT =10          4E
(sysce)         <110, xl10data, 0143>           EXT                   7D
(syscstk)       <110, xl10data, 0142>           EXT                   7C
(sysge)         <110, xl10data, 0141>           EXT                   7B
(sysgstk)       <110, xl10data, 0140>           EXT                   7A
```

```
< NLS, XL10DATA.NLS.15, >, 14-Jun-77 09:52 HGL ;;;;
FILE l10data % (arcsubsys,xl10,) (rel-nls,XL10DATA.rel,) %

SET TRACE=1; % 1 to compile trace globals, 0 to omit them %
% *** GLOBAL DATA FOR PAGE 0 *** %
% Special stuff declared in page 0, loaded at 140B %
   (signalsiz) CONSTANT=10; % number of nested signals allowed %      4A
   (catchsiz) CONSTANT=75; % number of catchphrases that can be invoked
   at once %                                                          4B
   (sigsiz) CONSTANT=12; % number of words in signal frame %          4C
   (catsiz) CONSTANT=10; % number of words in catch frame %           4D
   (stackdecsize) CONSTANT=10; % number of words in call stack
   descriptor %                                                       4E
   (gstksz) EXTERNAL CONSTANT =  4608;  %length of general call stack%
                                                                      4F
   (pstksz) EXTERNAL CONSTANT = 20; %length of pattern stack%         4G
%...KEEP THESE THINGS TOGETHER -- DONT ADD NEW DATA HERE...%
   DECLARE EXTERNAL %...important stuff...%
   %...user related data...%
      cinit,   % current initials of user %
   %...string analysis stuff...%
      flag = 1,       %the general flag%
      pstack[pstksz],     %pattern stack%
   %...editing global variables...%
      %...text editing global variables...%
         scndir,  % analyzer compiler scan direction %
         rplsid,  % psid of statement being replaced in sc %
         sptr1[2], sptr2[2],  % for append T-string %
         nsdbpt,  % byte pointer for statement being constructed%
   %...often used work areas...%
      swork, % string reading work area %
         swork1[6],
      stcwrk, % string construction work area %
         stcwr1[6],
   %...set command and aptstr routine...%
      modeset,       %  option set  %
      modeshift,     %  case set    %
      mkrstay,       %true if do not move markers%
      clmpty;   %reset value for clchng stack%
% accessed in statement construction%
   DECLARE EXTERNAL STACK clchng[30];
% general call stack + stack descriptor%
   (sysgstk) EXTERNAL [signalsiz*sigsiz]; % signal in progress stack %
                                                                      7A
   (sysge) EXTERNAL; % absolute end of signal stack %                 7B
   (syscstk) EXTERNAL [catchsiz*catsiz]; % catchphrase invoked stack %
                                                                      7C
   (sysce) EXTERNAL; % absolute end of catchphrase stack %            7D
   (gstkdesc) EXTERNAL [stackdecsize];                                7E
   (gstack) EXTERNAL [gstksz];                                        7F
% DECLARE's %
   DECLARE EXTERNAL STRING
      sar[2000];        % DO NOT USE -used by string primitives%
   DECLARE EXTERNAL
      %  signal info-- keep in this order!  do not add without changing
      runtime! %
```

```
    sysfrm, % current signaler's PORT ID %
    sysgco, % "" "" saved co-loc from his stack frame %
    syssig, % signal value%
    sysgtp, % signal type %
    sysg2, sysg3, sysg4, % signal arguments %
    syscat, % previous catchphrase pointer - for nesting signals;
    search starting location.  set in syshelp.  used in looping
    over catchphrase stack %
    sysbot, % stack pointer of bottom of the stack for signal
    propogation: may be sysbtm, or may be higher on the stack if
    the ange of propogation is to be restricted.  cf. systrm%
    syssstk, % stack pointer to bas of signalling stack %
    nestscan, % used in sysctn.  set in sysctn, popsig.  FALSE =>
    not a nested signal.  TRUE => nested signal. %
    sysgn, % signal in progress on current stack: also the number
    of the current signal since the top signal for a particular
    stack is the one in progress %
    tsysgn, % total number of signals in progress in system %
    sysinfo, % the address of the current signal on the signal
    stack %
  % catchphrase info %
    syscpm, % catchphrase parameter %
    systloc, % termiation location, global for systrm %
  % stack pointers to catchphrase stacks %
    sysctchall, % contains pointer to system catchphrase,
    syscatch, on catchphrase stack %
    syscpe, % pointer to end of catchphrase stack (dynamic end) %
    syscpt, % pointer to current catchphrase frame %
  % stack pointers to signal stacks %
    sysgpt, % current signal stack top pointer %
  % stack pointers to call stack(s) %
    pcstack, % stack pointer to bottom of "previous" current stack
    %
    sysbtm, % bottom stack frame ID for catch searches: a stack
    pointer to the bottom of the current stack %
    savebtm, % used to save sysbtm around calls on sigrestore %
    sysse, % pointer for (current) stack end, used by sysctn %
  % inter-procedure/coroutine communication globals %
    sysfctn, % TRUE if pcall comes from sysfctn (CONTINUE, SYSHLP,
    etc.); FALSE otherwise %
    sysbfg, % flag for signal routine - to set sysbot to restrict
    signal propogation.  TRUE only for NOTE(unwind or return).
    Otherwise false => use sysbtm.  Used in SYSHELP %
    syshcon, % gobal flag for syshelp/sysctn communication %
  %...general global variables...%
    linlink = 0, % stack pointer to gstack (if there is more than
    one stack) or 0 %
    syshlp, % port ID for HELP %
    sysabt, % port ID for ABORT %
    sysres, % port ID for RESUME %
    sysnot, % port ID for NOTE %
    sysinv, % port ID for INVOKE %
    syscon, % port ID for CONTINUE %
    % --- %
    startup, % REF for startup procedure %
    recover, % REF for recover procedure %
```

```
        % --- %
        syswhy, % global for runtime error reason string address %
        sysioc, % runtime error location %
        sysrip, % flag, "recover-in-progress" see sysrcv %
        sysetc, % error-type code
           (programbug, stkoverflow, uncaughtabort) %
        % --- %
        syszgn, % saved SIGNAL for uncaught ABORT %
        syszgt, % " for SIGNALTYPE %
        syszg2, syszg3, syszg4, % " for arguments %
      tempalsave, % to preserve al in pcall code %
      tempmregsave, % to preserve mreg in pcall code %
      srgsav[14], % to preserve registers 0-13 in pcall code %
      srg14, % to preserve register 14 in pcall code %
      srg15, % to preserve register 15 in pcall code %
      drpcat = 0,
      %+TRACE% % trace globals %                                          8B14
         systen=0, % TRACE ENABLE FLAG, set true to trace %
         systfg=0, % trace code flag, TRUE if trace in progress %
         systjfn, % TRACE.DATA file jfn , in left half %
         systpc, % trace.data file page count %
         systmd, % mode bits %
         systpn, % core page number to store trace data into %
         systpt, % ADBJN pointer into systpn %
         systck, % jobtm clock reading %
         systr1, % place to save R1 %
         systtp, % teletype output jfn/flag for trace %
         %+TRACE%                                                        9B14K
      %...other work areas...%
         ps, pe; %cells for BETWEEN construct%
   DECLARE EXTERNAL
      %collector sorter%
         lngflg; %true if length is considered befor value in sort%
   DECLARE EXTERNAL STACK
      btwstk[10];
   %...spec stack...%
      DECLARE EXTERNAL
         spsk = $spsk,     % spec stack pointer %
         spsk1= $spsk,     % initial pointer %
         b1[2],  b2[2],  b3[2],  b4[2],  b5[2],
         spskt = $b5;      % top of spec stack %
   %...correspondence list...%  %PASSED%
      DECLARE EXTERNAL
         clhead[4],  %header, see clhdr record%
         clstaddr,  %addr of allocated corres. list block%
         clsize,  %current size of clist block%
         clstad=$clhead;  %address of clist header%
% L10 LIST runtime globals %
   DECLARE EXTERNAL lstzone, blclst;
   DECLARE EXTERNAL STACK blcstk[10];
FINISH of l10data
```

```
< L10, L10LRP.NLS.4, >, 2-Dec-77 17:27 ANDY ;;;;
FILE l10lrp % (arcsubsys,xl10,) or (arcsubsys,l109,) to (L10,l10lrp,) %
   ALLOW!
% L10 LIST RUNTIME PACKAGE (part of L10RUNTIME) for PDP-10 %
% Internal Format %
   % The internal PDP-10 format of a list is:
      storwd
      list: XWD M,,L
      led1 (list element descriptor)
         ...
      ledM %
   % Where storwd is a runtime package word that is used to indicate if
   (and where) the list resides in allocated storage. A value of zero
   means that the list resides in declared storage or has not been
   referenced. In any case M designates the number of (following) words
   that may be used for elements.
      Another value storwd means
         The address of the list (itself in allocated storage).
         The original list.M stays even tho the max is larger. It is in
         the allocated list.M The list.L and the allocated list.L are
         the same and indicate the length.
      %
% Definitions %
   REF biclst; % declared in l10data %
   (ledr) RECORD % PDP-10 list element descriptor%                    4B
      ledval [18], %address/value of element%
      ledubv[ ledtyp[3], ledbits[3] ], % type and user bits %
      ledalo  [1]; %element space allocated if TRUE%
   DECLARE EXTERNAL CONSTANT % descriptor types %
      %
      lnull  = 0 ,   NULL
      linteg = 2 ,   INTEGER
      lstrin = 3 ,   STRING
      llist  = 4 ,   LIST
      lblock = 1 ,   BLOCK
      %
      ldescr = 101 , % DESCRIPTOR -- never stored as type %
      lnewde = 100 ; % NEWDESCRIPTOR -- never stored as type  %
   DECLARE CONSTANT
      nulldescr = 0B;
      % ledval = 0 %
      % ledtyp = 0 %
      % ledalo = 0 %
   DECLARE CONSTANT
      blkadj=0, % result of getblk plus blkadj is first free word in
      block %
      lsthdr=1, % number of words in list header (stowrd) %
      storwd=18M, % index to get storwd %
      listmx = 120, % 2 times max list size (= 60 ) %
      blcsiz=30; % initial size of construction workspace %
(nulist) % set a list or sublist to NULL %                            5
   PROCEDURE (
      list REF, % address of list %
      sublist, % boolean: TRUE if e1,e2 or FALSE if entire list %
      e1, e2) ; % nullify e1 thru e2 if sublist=TRUE %
   % If "sublist" is FALSE, effectively sets list.L to zero.
```

```
   Otherwise, discards elements "e1" through "e2", shifing forward any
   elements behind them. Written so nulist(list,FALSE) will work !! %
   LOCAL
      i, % element index %
      nlength, % new list length %
      alist REF, % allocated list address %
      top, % highest elelement to delete %
      bot; % lowest %
   IF list.L =0 THEN RETURN; % don't waste time %
   IF sublist THEN BEGIN bot _ MAX(1,e1); top _ MIN(e2,list.L); END
   ELSE BEGIN bot _ 1; top _ list.L; END;
   nlength _ bot-1; % number of elements in lower segment %
   % free all allocated storage in the elements to be eliminated %
   FOR i _ bot UP UNTIL > top DO
      IF top < list.L THEN % move down top elements %
         BEGIN
         wrielm($list, lnewde, rdlelm($list, TRUE, top+1), TRUE, i);
         BUMP top;
         BUMP nlength; % one more real element %
         END
      ELSE
         BEGIN
         IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
         freelm(alist[i]); % free elements %
         END;
   list.L _ nlength;
   IF list[storwd] # 0 THEN % allocated list to take care of %
      BEGIN
      &alist _ list[storwd];
      alist.L _ nlength; % set length both places %
      lstrst($list, $alist); % move back if possible %
      END;
   RETURN;
   END.
(rdlelm) % read list element, return descriptor, element value %        6
   PROCEDURE (
      list REF,  % address of list %
      delete, % boolean: delete element if true %
      index % list element index % ) ;
   % Returns the descriptor and value (descr, value) for (addr of L10
   string, list, or block; or integer) of element "index" of list
   "list".  If "delete" is TRUE, the source element descriptor is
   replaced with a null descriptor. The element itself is left. This is
   how a "MOVE" is done. %
   LOCAL
      descr, % the descriptor %
      value, % the value %
      alist REF; % the address of list elements %
   IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
   IF index NOT IN [1,alist.L] THEN RETURN(nulldescr,0);
   descr _ alist[index] ;
   IF descr.ledtyp = linteg AND descr.ledalo THEN
      value _ [descr.ledval]
   ELSE
      value _ descr.ledval ;
   IF delete THEN
```

```
      alist[index] _ nulldescr;
   RETURN ( descr, value );
   END.
(wrlelm) % write list element %                                        7
   PROCEDURE (
      list REF, % list address %
      type, % shows what type 'value' is %
      value, % the value itself %
      replace, % boolean: TRUE= store in list[index], FALSE= append %
      index) ; % list element index %
   % If "replace" is FALSE, appends element of type "type" and value
   "value" to list "list".  Otherwise, replaces element "index" with
   it. Written so that the fifth argument (index) may be omitted if
   replace=FALSE  %
   LOCAL
      alist REF, % address of list elements %
      i, % list element index %
      size, % size of new list %
      myindex;  % real list element index %
         % cannot touch index if not provided ! %
   IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
   CASE replace OF
      = FALSE: % not replacing an element %
        BEGIN
           IF alist.L >= alist.M THEN
              BEGIN
              IF (size _ MAX(blcsiz, list.L*2) ) > listmx THEN
                 sysrcv(programbug, $"excessive list overflow",
                 sysclr());
              CASE &list OF
                 = &blclst: % update &list after len changes %
                    BEGIN
                    setllen( &list, size);
                    &list _ &blclst;
                    END;
                 ENDCASE setllen(&list, size);
              IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
              END;
           BUMP alist.L ;
           IF &alist # &list THEN list.L _ alist.L;
           myindex _ alist.L ;
        END;
      ENDCASE
        BEGIN
        CASE index OF
           <= list.L: % replacing an existing element %
              BEGIN
              freelm (alist[index]) ;
              myindex _ index;
              END ;
           ENDCASE % extend list up to this element and then append %
              BEGIN
              FOR i_list.L+1 UP UNTIL >= index DO
                 wrlelm($list, lnull, 0, FALSE);
              IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
                 % alist may have changed %
```

```
            REPEAT CASE 2 ( FALSE); % do the append %
            END;
        END ;
    alist[myindex] _ makelm ( type, value ) ;
    RETURN;
    END.
(setllen) % set list (allocated) length %                                   8
    PROCEDURE(
        list REF, % list address %
        size); % desired size %
    LOCAL
        i, % list element index %
        alist REF,
        nlist REF; % for new list %
    % makes sure that list has max length 'size' %
    IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
    IF alist.M < size THEN
        BEGIN
        &nlist _ getlst(size);
        nlist.L _ alist.L;
        FOR i_1 UP UNTIL > alist.L DO
            nlist[i] _ alist[i];
        list[storwd] _ &nlist; % fix up stowrd %
        CASE &alist OF
            =&blclst: % from construction workspace %
                BEGIN
                frelst(&blclst);
                &blclst _ &nlist;
                END;
            = &list: NULL; % just moved to alocated sto %
            ENDCASE
                frelst($alist);
        END;
    RETURN;
    END.
(blc) % begin list construction (initialization) %                          9
    PROCEDURE;
    % push current list construction state and begin a new list. The
    list is built in allocated storage, but may not stay there. See
    (elc). %
    PUSH &blclst ON blcstk;
    &blclst _ getlst(blcsiz);
    blclst.L _ 0;
    RETURN;
    END.
(aplelm) % append list element (to construction workspace) %               10
    PROCEDURE(
        type, % shows what type 'value' is %
        value); % the value to be stored %
    % Appends element of type "type" and value "value" to list in list
    construction workspace. %
    wrlelm($blclst, type, value, FALSE);
    RETURN;
    END.
(apsubl) % append sublist (or entire list) to construction workspace %
                                                                           11
```

```
    PROCEDURE(
       flags, % determines mode: see below for meaning %
       list REF, % address of source list %
       e1, e2); % use list[e1] thru list[e2] %
    % flags
         bit 35
            0 - append only elements e1 through e2 of list to the
          worklist
            1 - append all elements of list to the worklist
         bit 34
            0 -
            1 - move
         bit 33
            0 -
            1 - copy
         bit 32
            0 -
            1 - values of source elements are stored as integers
       %
    % appends the sublist e1 thru e2, or the entire list, to the
    construction workspace. Flags indicate copy/move and sublist/all
    modes. %
    LOCAL
       descr, value, % for element begin appended %
       top, % last element to append %
       i; % list element index %
    IF flags .A 1 THEN % means append whole list %
       BEGIN
       i _ 1;
       top _ list.L;
       END
    ELSE
       BEGIN
       i _ MAX(1,e1);
       top _ MIN(list.L, e2);
       END;
    FOR i UP UNTIL > top DO
       BEGIN
       descr _ rdlelm($list, flags .A 2 % move %, i: value);
       wrlelm( $blclst, IF flags .A 8 THEN linteg ELSE IF flags .A 2
       THEN lnewde ELSE ldescr, IF flags .A 8 THEN value ELSE descr,
       FALSE);
       END;
    RETURN;
    END.
(elc) % end list construction and store list somewhere %                12
    PROCEDURE (list REF); % list address or zero %
    % If argument "list" is present (i.e. non-zero), stores list in list
    construction workspace as list "list" and releases the workspace.
    Otherwise, simply returns addr "worklist" of and responsibility for
    list in workspace. %
    LOCAL
       descr, % a new descriptor %
       i; % list element index %
    descr _ 0;
    CASE &list OF
```

```
      =0: % just return the workspace address %
         BEGIN
         descr.ledval _ &blclst;
         descr.ledalo _ TRUE;
         END;
      ENDCASE
         BEGIN
         list.L _ blclst.L;
         lstrst($list, $blclst); % return list if possible %
         descr.ledval _ &list;
         END;
   POP blcstk TO &blclst;
   descr.ledtyp _ llist;
   RETURN(descr);
   END.
(lstrst) % restore list : return to declared storage if possible %     13
   PROCEDURE (
      list REF, % declared list %
      alist REF); % allocated list %
   % move the list in 'alist' to 'list' if it will fit. if moved,
   release the storage for 'alist' %
   LOCAL
      i; % list element index %
   CASE list.M OF
      >= alist.L: % can move it %
         BEGIN
         list[storwd] _ 0;
         FOR i_1 UP UNTIL > alist.L DO
            list[i] _ alist[i];
         freist(&alist);
         END;
      ENDCASE list[storwd] _ $alist; % point to it %
   RETURN;
   END.
(makelm) % make new list element given type and value %                14
   PROCEDURE (
      type, % shows type of value provided %
      value REF ) ; % the value depends on type, see case stmt. %
   % makelm makes a descriptor for a new list element, including
   allocating storage and copying data, and returns it. %
   LOCAL
      descr_0, % the result %
      i, % list element index %
      size, blkadr; % size and address of block obtained %
   CASE type OF
      = lnewde: % value is a new descriptor, just use it if ok %
         IF ckdesc ( &value ) THEN descr _ &value
         ELSE sysrcv(programbug, $"bad new-descriptor for list,
         makelm", sysclr());
      = lnull: % value ignored. he wants a null element %
         descr _ nulldescr  ;
      = ldescr: % value is existing element descr. copy it %
         BEGIN
         descr _
            makelm ( &value.ledtyp, IF &value.ledtyp = linteg AND
            &value.ledalo THEN [&value.ledval] ELSE &value.ledval) ;
```

```
        descr.ledbits _ &value.ledbits;
        END;
      = linteg: % value is 36 bit integer. check size etc. %
        IF &value NOT IN [0, 18M] THEN
           BEGIN
           blkadr _ getblk ( 1, lstzone ) ;
           IF blkadr=0 THEN ABORT(listspace, $"exceed LIST allocation
           zone");
           [blkadr+blkadj] _ &value ;
           descr.ledval _ blkadr + blkadj ;
           descr.ledtyp _ linteg ;
           descr.ledalo _ TRUE ;
           END
        ELSE
           BEGIN
           descr.ledval _ &value ;
           descr.ledtyp _ linteg ;
           descr.ledalo _ FALSE ;
           END ;
      = lstrin: % value is string address. copy the string %
        BEGIN
        blkadr _ getstring(value.L, lstzone);
        *[blkadr]* _ *value* ;
        descr.ledval _ blkadr ;
        descr.ledtyp _ lstrin ;
        descr.ledalo _ TRUE ;
        END ;
      = llist: % value is list address. copy entire list!! %
        BEGIN
        blkadr _ getlst(value.L) ;
        [blkadr].L _ value.L;
        FOR i _ 1 UP UNTIL > value.L  DO
           [blkadr+i] _ makelm ( ldescr, value[i] ) ;
        descr.ledval _ blkadr;
        descr.ledtyp _ llist ;
        descr.ledalo _ TRUE ;
        END ;
      = lblock: % value is block address (adr of word with size) copy %
        BEGIN
        size _ value[-1].blklength -1 ;
        blkadr _ getblk ( size, lstzone ) ;
        IF blkadr=0 THEN ABORT(listspace, $"exceed LIST allocation
        zone");
        blkadr _ blkadr + blkadj;
        FOR i _ 0 UP UNTIL >= size DO
           [blkadr+i] _ value[i] ;
        descr.ledval _ blkadr ;
        descr.ledtyp _ lblock ;
        descr.ledalo _ TRUE ;
        END ;
    ENDCASE sysrcv(programbug, $"bad type given to makelm", sysclr());
    RETURN ( descr ) ;
    END.
(freelm) % free any storage associated with given list element %        15
    PROCEDURE ( descr ) ; % the list element descriptor %
    % frees any allocated storage associated with this element %
```

```
    LOCAL
       blkadr; % address of block to free %
    IF descr.ledalo THEN
       CASE descr.ledtyp OF
          = linteg:
             BEGIN
             blkadr _ descr.ledval - blkadj ;
             IF freeblk ( blkadr, lstzone ) =0 THEN sysrcv(programbug,
             $"cannot deallocate a list", sysclr());
             END ;
          = lstrin:
             BEGIN
             blkadr _ descr.ledval ;
             freestring(blkadr, lstzone);
             END ;
          = llist:
             BEGIN
             nulist ( descr.ledval, FALSE ) ;
             frelst(descr.ledval);
             END;
          = lblock:
             BEGIN
             blkadr _ descr.ledval ;
             IF freeblk ( blkadr-blkadj, lstzone ) =0 THEN
             sysrcv(programbug, $"cannot deallocate a list", sysclr());
             END ;
          ENDCASE sysrcv(programbug, $"bad type given to freelm",
          sysclr());
    RETURN(nulldescr);
    END.
(ckdesc) % check list element descr for reasonable values %                16
    PROCEDURE ( descr ) ;
    %ckdesc checks a list element descriptor to see that it meets some
    criteria, for example that the type is in range and the flag bits
    are reasonable. ckdesc returns true if all is well, false otherwise.
    %
    IF descr.ledtyp NOT IN [inull, llist] THEN RETURN ( FALSE ) ;
    %one could check addðress range depending on local or global
    declared vs allocated storage. %
    % one could ckeck to see that no unused bits are on. %
    RETURN(TRUE);
    END.
(getlst) % get a storage block for a list %                                17
    PROCEDURE (size); % number of list elements, max %
    % Uses getblk to obtain block of storage 'size'+1 words long %
    LOCAL adr REF; % address %
    &adr _ getblk( size+1 + lsthdr, lstzone);
    IF &adr=0 THEN ABORT(listspace, $"exceed LIST allocation zone");
    &adr _ &adr + blkadj + lsthdr;
    adr.M _ size;
    RETURN(&adr);
    END.
(frelst) % free an allocated list element %                                18
    PROCEDURE (list REF); % address of list %
    % Uses freblk to free up the allocated block of storage %
    IF freeblk( &list - lsthdr - blkadj, lstzone) =0 THEN
```

```
    sysrcv(programbug, $"cannot deallocate a list", sysclr());
    RETURN;
    END.
(lreadb) % read user bits from list element descr %                    19
    PROCEDURE(
        list REF, % list adress %
        index); % element index %
    LOCAL
        descr,
        alist REF;
    % return composite type, user bits %
    IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
    IF index NOT IN [1,alist.L] THEN RETURN(0,0);
    descr _ alist[index];
    RETURN(descr.ledubv, descr.ledbits);
    END.
(lsetb) % set user bits in list elment descr %                         20
    PROCEDURE(
        list REF, % lis address %
        index, % element index %
        value); % new user-bits value %
    % stores new value in user bits, return TRUE if OK %
    LOCAL
        alist REF;
    IF (&alist _ list[storwd]) = 0 THEN &alist _ &list;
    IF index NOT IN [1,alist.L] THEN RETURN(FALSE);
    alist[index].ledbits _ value;
    RETURN(TRUE);
    END.
(sysfll) %Free Local Lists. Call produced by "NULL-LISTS" %            21
    PROCEDURE(
        ptr REF); % address of array of local lists %
    % L10 compiler produces a call to sysfll for NULL-LISTS statement.
    The argument is the address of an array of N relative list locations
    in the calling procdure. All those lists are nulled. A zro ends the
    array. %
    WHILE ptr#0 DO
        BEGIN
        nulist( ptr.RH + [N].RH, FALSE);
        BUMP &ptr;
        END;
    RETURN;
    END.
FINISH L10LRP
```

```
((stast) RECORD stadr) <110, xl10runtime, 01422> FIELD - 18            7B4
((stbw) RECORD stwc) <110, xl10runtime, 01420>   FIELD - 9             7B2
(alostk)        <110, xl10runtime, 03541>        PROCEDURE             8J1
(apachr)        <110, xl10runtime, 0289>         LOCAL                 9D
(apblnk)        <110, xl10runtime, 0705>         PROCEDURE             11N
(apchr)         <110, xl10runtime, 01662>        LOCAL                 11H
(apstore)       <110, xl10runtime, 0598>         LOCAL                 11H2D
(aptstr)        <110, xl10runtime, 0294>         LOCAL                 9E
(ascom)         <110, xl10runtime, 0608>         PROCEDURE             11J
(asrref)        <110, xl10runtime, 0548>         LOCAL                 11A
(astruc)        <110, xl10runtime, 01249>        PROCEDURE             11V
(bfs)           <110, xl10runtime, 0997>         LOCAL                 12E
(bits)          <110, xl10runtime, 03920>        EXT                   6E
(bptctn)        <110, xl10runtime, 03534>        LOCAL                 8E2E7
(bptinv)        <110, xl10runtime, 02284>        LOCAL                 8E4F3
(bptrcv)        <110, xl10runtime, 02294>        LOCAL                 8B2
(bptres)        <110, xl10runtime, 02285>        LOCAL                 8E3E8
(bptrtn)        <110, xl10runtime, 02403>        LOCAL                 8E4H8E
(bpttrm)        <110, xl10runtime, 02259>        LOCAL                 8E5F
(bsc)           <110, xl10runtime, 0165>         LOCAL                 9A
(byteptr)       <110, xl10runtime, 01520>        LOCAL                 7A
(callstksize)   <110, xl10runtime, 03995>        EXT CONSTANT =2000B   8A4A
(catch)         <110, xl10runtime, 02149>        CATCHPHRASE           9E21
(catchnam)      <110, xl10runtime, 01613>        LOCAL                 8E7A1
(catchnam)      <110, xl10runtime, 01606>        LOCAL                 8E6A1
(catchname)     <110, xl10runtime, 01625>        LOCAL                 8F3A1
(catchname)     <110, xl10runtime, 02263>        LOCAL                 8F2A1
(catenc)        <110, xl10runtime, 02045>        CONSTANT =2           8A5C
(cattrm)        <110, xl10runtime, 02044>        CONSTANT =1           8A5B
(catloc)        <110, xl10runtime, 02043>        CONSTANT =0           8A5A
(catmrk)        <110, xl10runtime, 03633>        CONSTANT =8           8A5I
(catown)        <110, xl10runtime, 02317>        CONSTANT =7           8A5H
(catp)          <110, xl10runtime, 03847>        REF                   8F4C
(catprm)        <110, xl10runtime, 02308>        CONSTANT =6           8A5G
(catrtn)        <110, xl10runtime, 02047>        CONSTANT =4           8A5E
(cats)          <110, xl10runtime, 02046>        CONSTANT =3           8A5D
(catsiz)        <110, xl10runtime, 02029>        CONSTANT =10          8A5L
(catstk)        <110, xl10runtime, 03634>        CONSTANT =9           8A5J
(cattloc)       <110, xl10runtime, 02048>        CONSTANT =5           8A5F
(cct)           <110, xl10runtime, 01115>        PROCEDURE             12O
(charno)        <110, xl10runtime, 01667>        LOCAL                 11G1A
(chbpt1)        <110, xl10runtime, 0588>         LOCAL                 11G2G
(chbptr)        <110, xl10runtime, 01660>        LOCAL                 11G
(clbuff)        <110, xl10runtime, 01436>        FIELD - 18            7D5
(clcc1)         <110, xl10runtime, 01427>        FIELD - 12            7C3
(clcc2)         <110, xl10runtime, 01428>        FIELD - 12            7C4
(clcnt)         <110, xl10runtime, 01432>        FIELD - 9             7D1
(cldsr)         <110, xl10runtime, 01490>        LOCAL                 9F
(clfixed)       <110, xl10runtime, 01429>        FIELD - 1             7C5
(clino1)        <110, xl10runtime, 01433>        FIELD - 7             7D2
(clino2)        <110, xl10runtime, 01434>        FIELD - 7             7D3
(clndr)         <110, xl10runtime, 01431>        RECORD                7D
(clst1)         <110, xl10runtime, 01425>        FIELD - 36            7C1
(clst2)         <110, xl10runtime, 01426>        FIELD - 36            7C2
(cltype)        <110, xl10runtime, 01435>        FIELD - 6             7D4
(cptse)         <110, xl10runtime, 01137>        PROCEDURE             12S
```

(sysclr)	<110, xl10runtime, 02178>	LOCAL	8H3
(syscte)	<110, xl10runtime, 01594>	LOCAL	8B7D
(sysctn)	<110, xl10runtime, 03482>	LOCAL	8E2
(sysdis)	<110, xl10runtime, 01611>	LOCAL	8E7
(sysdp1)	<110, xl10runtime, 02233>	LOCAL	8F5
(sysdpall)	<110, xl10runtime, 03830>	LOCAL	8F4
(sysdrp)	<110, xl10runtime, 01623>	LOCAL	8F3
(sysena)	<110, xl10runtime, 01604>	LOCAL	8E6
(sysenc)	<110, xl10runtime, 02260>	LOCAL	8F2
(sysent)	<110, xl10runtime, 02205>	LOCAL	8B6A
(syshelp)	<110, xl10runtime, 01797>	LOCAL	8E1
(sysnxp)	<110, xl10runtime, 01586>	LOCAL	8B7B
(sysovr)	<110, xl10runtime, 01782>	LOCAL	8B6B
(syspc1)	<110, xl10runtime, 03647>	LOCAL	8B6D9
(syspc2)	<110, xl10runtime, 03608>	LOCAL	8B6D12
(syspc3)	<110, xl10runtime, 03658>	LOCAL	8B6D12B
(syspc4)	<110, xl10runtime, 04162>	LOCAL	8B6D12C5(
(syspop)	<110, xl10runtime, 01936>	LOCAL	8E4H
(sysprc)	<110, xl10runtime, 02092>	LOCAL	8H1
(sysrcv)	<110, xl10runtime, 01569>	LOCAL	8B
(sysrsume)	<110, xl10runtime, 01794>	LOCAL	8E3
(sysrtf)	<110, xl10runtime, 01572>	LOCAL	8B5C
(sysrtn)	<110, xl10runtime, 01573>	LOCAL	8B5B
(sysrtt)	<110, xl10runtime, 01570>	LOCAL	8B5A
(syssys)	<110, xl10runtime, 01711>	LOCAL	8C
(syster)	<110, xl10runtime, 01733>	LOCAL	8B7E
(systi1)	<110, xl10runtime, 03319>	LOCAL	8D1E2
(systi2)	<110, xl10runtime, 03328>	LOCAL	8D1E3
(systi3)	<110, xl10runtime, 03329>	LOCAL	8D1E4
(systi4)	<110, xl10runtime, 03327>	LOCAL	8D1E5
(systi5)	<110, xl10runtime, 03330>	LOCAL	8D1E6
(systj3)	<110, xl10runtime, 03318>	LOCAL	8D1E1
(systoff)	<110, xl10runtime, 03375>	LOCAL	8D2
(systrc)	<110, xl10runtime, 03296>	LOCAL	8D1
(systrm)	<110, xl10runtime, 01628>	LOCAL	8E5
(systwd)	<110, xl10runtime, 03331>	LOCAL	8D1E7
(systz1)	<110, xl10runtime, 01576>	LOCAL	8B5B4
(systz2)	<110, xl10runtime, 01580>	LOCAL	8B5C5
(systz3)	<110, xl10runtime, 02208>	LOCAL	8B6A3
(systz4)	<110, xl10runtime, 02402>	LOCAL	8B6C4
(systz5)	<110, xl10runtime, 02218>	LOCAL	8B6D10
(sysufl)	<110, xl10runtime, 01780>	LOCAL	8B7F
(sysvoke)	<110, xl10runtime, 01795>	LOCAL	8E4
(tcatp)	<110, xl10runtime, 03795>	REF	8C2A
(temp)	<110, xl10runtime, 03861>	LOCAL	8G3C
(teststk)	<110, xl10runtime, 04184>	LOCAL	8H1D
(tsigptr)	<110, xl10runtime, 03862>	REF	8G3E

```
< L10, XL10RUNTIME.NLS.3, >, 28-Oct-77 16:37 DIA ;;;;.MCS=(13-May-77
13:00);.MCSTPos=RightMark;
FILE xl10runtime  % (arcsubsys,XL10,) to (rel-nls, XL10RUNTIME,) %
ALLOW! % system interface stuff here %
SET TRACE=1; % 1 to compile trace code, 0  to omit it %

% NOTES ABOUT RUNNING STANDALONE L10 PROGRAMS %
   % the following files should be loaded with your program to generate
   a standalone l10 program:
      <rel-nls>l10data.rel       - l10 data package
      <rel-nls>l10runtime        - l10 runtime package
      <subsys>stenex             - tenex jsys definitions
   The following procedures must be provided:
      A startup procedure
         This procedure is given control after the L10 environment is
         established.  The address of this procedure must be stored in
         system variable "startup". No arguments are passed.
      A recovery procedure
         This procedure is given control when the runtime package
         encounters an error and cannot contnue.  At the entry point,
         the stack has been totally reset, but non-system globals
         remain unchanged from their condition at the time of the
         error.  Three arguments are passed: the first indicates the
         type of error and the second is the address of a string
         describing the error and the third provides an address to help
         locate the offending code. The address of this procedure must
         be stored in system variable "recover".
         The last two arguments may be passed to procedure "errmsg" in
         order to get a user-readable error string printed.  The
         arguments are: (jfn, arg2, arg3) where arg2 and arg3 are the
         arguments passed to the recover routine.
   the following will be undefined unless they are provided by the
   user:
      filesc
         this procedure is called from procedure esc (end string
         construction) when esc detects that the strings being dealt
         with do not have the stastr bit set (i.e., they are not
         astrings)
         if a user is not dealing with any strings other than literal,
         local, or global strings (for example, strings within a data
         file), then this procedure need not be supplied.
         to see how NLS handles this see <nls, utilty, filesc>
      flcpfse
         this procedure is called from procedure cpfse when cpfse
         detects that the strings being dealt with do not have the
         stastr bit set (i.e., they are not astrings)
         if a user is not dealing with any strings other than literal,
         local, or global strings (for example, strings within a data
         file), then this procedure need not be supplied.
         to see how NLS handles this see <nls, utilty, flcpfse>
      flfechc1
         this procedure is called from procedure fechc1 when fechc1
         detects that the strings being dealt with do not have the
         stastr bit set (i.e., they are not astrings)
         if a user is not dealing with any strings other than literal,
         local, or global strings (for example, strings within a data
```

tile), then this procedure need not be supplied.
to see how NLS handles this see <nls, utilty, flfechcl>
The starting point for the program is the procedure whose address is in system variable "startup". However, execution must start at location l10start in order to initialize the program environment.
To run a program, load it and go into DDT. Then place the address of the starting procedure in system variable "startup", the address of the recover procedure in location "recover", and start executing at location "l10start".
To make use of the TENEX entry vector, do one of the following:
1) Set "startup" to the address of an initializing procedure that will set the entry vector to a table of JRST instructions and then HALTF. For each entry point, the JRST goes to a code branch that sets "startup" to the desired starting procedure and then does a GOTO location "l10start".
Note that, while executing the code branches that set the "startup" location, the runtime environment is not intitialized. Therefore it is impossible to do procedure calls or reference locals. However, a simple assignment such as the following is OK:
startup _ $procedure;
Set "recover" to the desired procedure address and start executing at "l10start" and when your program halts, save the core immage. At startup time, the proper code branch will set the "startup" location and pass control to L10 initialization which will pass control to the startup procedure.
2) If your program does no initialization before you save it, it may be desirable to prepair the entry vector as above and use the EXEC's ENTVEC command to establish your entry vector.
Load the program, setup "recover", go to the EXEC and do the ENTVEC comand and then save the core immage.

if you have problems or comments getting standalone L10 programs to the point where the bugs are in your program and not associated with the L10 environment, contact a systems programmer at SRI-ARC
.Pes;%

```
% DECLARATIONS %
  REGISTER
     a1 = 10, %working register%
     a2 = 11, %working register%
     a3 = 12, %working register%
     a4 = 13, %working register%
     r1 = 1,  %working register%
     r2 = 2,  %working register%
     r3 = 3,  %working register%
     r4 = 4,  %working register%
     r5 = 5,  %working register%
     rp = 9, %record pointer%
     p = 7,   %pattern stack%
     wa = 8,   %work area stack%
     mreg = 6, % "meta" result register, success/fail and port %
     s = 15,   %general call stack pointer%
     m = 14;  %mark stack pointer%
  DECLARE EXTERNAL errmsbase = 140B;
  EXTERNAL
     readc,      %entry to READC routine%
     l10start; % entry to initialization code %
  DECLARE EXTERNAL
     empty = 0, endfil = -1,
     origin = 2 %=fbhdl%, nullch = 0,
     chbmty = 440700000001B;
        % $"xxx"+chbmty is byte pointer to first byte %
  (bits) EXTERNAL = ( % 36 bit table: bits[n] is a mask with 1 in the
  nth bit from the left; bits[0] is the leftmost bit %                6E
     4B11, 2B11, 1B11,
     4B10, 2B10, 1B10,
     4B9, 2B9, 1B9,
     4B8, 2B8, 1B8,
     4B7, 2B7, 1B7,
     4B6, 2B6, 1B6,
     4B5, 2B5, 1B5,
     4B4, 2B4, 1B4,
     4B3, 2B3, 1B3,
     4B2, 2B2, 1B2,
     4B1, 2B1, 1B1,
     4, 2, 1 );
  DECLARE EXTERNAL
     sdbhdl=5;    %length of sdb header%
  DECLARE EXTERNAL
     forward = 1; % used by READC constructs %
  %.Pes;%
```

```
% RECORDS %
  (byteptr)    % field definitions for PDP-10 byte pointer word %     7A
     RECORD
     bpadr[18], bpindx[6], bpsize[6], bpbitpos[6];

  (stidr) RECORD stpsid[18], stfile[8], stastr[1];                    7B
     % -- (filmnp, lodrng) and (filmnp, lodsdb) cheat and make use of
     the format of an stid -- so please don't change this without
     checking those routines and telling WHP %
     (stbw) RECORD stwc[9], stblk[9];                                 7B2
     (stsdb) RECORD stpsdb[18];                                       7B3
     (stast) RECORD stadr[18];                                        7B4
     (stsidr) RECORD stsidf[6], stsid[30];
                                                                      7B5
  (clistr) RECORD %clist entry%                                       7C
     clst1[36],  %original or updated stid%
     clst2[36],  %successor stid%
     clcc1[12],  %character count for first%
     clcc2[12],  %character count for second%
     clfixed[1]; %for aptstr%

     DECLARE EXTERNAL cll = 3, clmax = 50;

  (clhdr) RECORD %clist header%                                       7D
     clcnt[9],   %count of entries in use%
     clfno1[7],  %type used to generate clist%
     clfno2[7],  %type used to generate clist%
     cltype[6],  %type used to generate clist%
     clbuff[18]; %address of buffer%

  (ring) RECORD % *** ringl is length%                                7E
     rsub[18],     %psid of sub of this statment%
     rsuc[18],     %psid of suc of this statement%
     rsdb[18],     %psdb of sdb for this statement%
     rinst1[7],    %DEX interpolation string-- scratch space%
     rinst2[7],    %DEX interpolation string-- scratch space%
     rdummy[1],    %DEX dummy flag-- scratch space%
     repet[3],     %DEX repetition-- scratch space%
     rhf[1],       %head flag. true if this is head of plex%
     rtf[1],       %tail flag, true if tail of plex%
     rname1[1],    %name flag, true if statement has a name%
     rtorgin[1],   %inferior tree origin flag, true if origin%
     rnull[1],     %unused%
     rnameh[30],   %name hash for this statement%
     rsid[30];     %statement identifier%
     %although only need four words, use five so that have room
     to grow%

  (sdbhead) RECORD % *** sdbhdl is length%                            7F
     sgarb[1],     %true if this sdb is garbage%
     slength[9],   %number of words in this sdb%
     schars[11],   %number of characters in this statement%
     slnmd1[7],    %left name delimiter for statement%
     srnmd1[7],    %right name delimiter for statement%
     spsid[18],    %psid of the statement for this sdb%
     sname[11],    %position of character after name%
```

```
   stime[36],   %date and time when this sdb created%
   sinit[21],   %initials of user who created this sdb%
   sptype[15],  %property type of this data block%
      %
      txttyp   = text data block (SDB)
      dhdtyp   = diagram header block
      segtyp   = segment data block
      %
   spsdb[18],   %PSDB of the next property data block 0=tail%
   sitpsid[18]; %PSID to head of inferior tree if any%
   %sgarb and slength must be in the first word of the header
   for newsdb%

(frmtctrl) RECORD %format control for "STRING" construct %          7G
   fmncols[7],          % # columns (with punctuation) / 0: as many as
   needed %
   fmjstfy[1],          % TRUE: right justify; FALSE: left justify %
   fmfill[1],           % fill field with   TRUE: 0s;  FALSE: spaces %
   fmigcl[1],           % TRUE: treat as 36 bit unsigned quantity %
   fmsgne[1],           % TRUE: ignore LH of value and extend sign of
   RH %
   fmpsgn[1],           % TRUE: print + if val > 0 / val is 36 bit
   quantity %
   fmdecm[1],           % TRUE: print terminating decimal point %
   fmovrf[3],           % column overflow control
      0 - print nothing
      1 - print as many most significant digits (MSDs) as fit
      2 - print as many least significant digits (LSDs) as fit
      3 - print blanks
      4 - print astericks in entire field
      5 - print as many MSDs as fit followed by one asterick
      6 - print as many LSDs as fit preceded by one asterick %
   fmfloat[20];         % reserved for eventual floating point format
   control %

%.Pes;%
```

```
% system startup, returns, signal and fatal error code & declarations %
  % DECLARATIONS for CALL/RETURN mechanisms %
    EXTERNAL % labels (for runtime "procedures") %
      bptrcv, bptctn, bptres, bptirv, bpttrm, bptrtn,
      sysrtn, sysrtt, sysrtf, syster,
      sysnxp, syscer, syscte, sysufl,
      sysovr, syspop, sysent, syscent, pcall;
    EXTERNAL % labels for trace package %
      systz1, systz2, systz3, systz4, systz5, systil, systi2,
      systi3, systi4, systi5, systwd; % trace labels %
    % special code for enable count %
      (dropped) CONSTANT=-100000; % this enable count means
      catchphrase dropped %                                          8A3A
    % size of allocated stack %
      (callstksize) EXTERNAL CONSTANT = 2000B;                       8A4A
    % catchphrase frame indices %
      (catloc) CONSTANT=0; % location of catchphrase %               8A5A
      (catfrm) CONSTANT=1; % (frame) PORT id for catchphrase %       8A5B
      (catenc) CONSTANT=2; % enable count for catchphrase %          8A5C
      (cats) CONSTANT=3; % S at invoke time %                        8A5D
      (catrtn) CONSTANT=4; % real return location for owning
      procedure %                                                    8A5E
        % zero if not first invoked catchphrase owned by procedure
        %
      (cattloc) CONSTANT=5; % termination location %                 8A5F
      (catprm) CONSTANT=6; % single catchphrase parameter %          8A5G
      (catown) CONSTANT=7; % (frame) PORT id for owning procedure %
                                                                     8A5H
      (catmrk) CONSTANT=8; % indicates which signals this
      catchphrase has seen %                                         8A5I
      (catstk) CONSTANT=9; % stack id of stack which owns this
      catchphrase %                                                  8A5J
      % ------- %
      (catsiz) CONSTANT=10; % entries in catch frame %               8A5L
    % stack descriptor indices %
      (stkmsave) CONSTANT = -1; % saved M ptr %                      8A6A
      (stkssave) CONSTANT = -2; % saved S ptr %                      8A6B
      (stklink) CONSTANT = -3; % forward link to next stk or 0 %     8A6C
      (stkblink) CONSTANT = -4; % backward link to previous stk or
      0%                                                             8A6D
      (stkzone) CONSTANT = -5; % allocation zone if allocated stack;
      0 otherwise%                                                   8A6E
      (stksize) CONSTANT = -6; % size of usable stack size of usable
      stack area%                                                    8A6F
      (stksig) CONSTANT = -7; % Number of the current signal in
      progress on this stack.%                                       8A6G
      (stkdalo) CONSTANT = -8; % Stack which is deallocating this
      stack: 0 except in dalostk%                                    8A6H
      (stkmdalo) CONSTANT = -9; % M for Stack which is deallocating
      this stack: 0 except in dalostk%                               8A6I
      (stksdalo) CONSTANT = -10; % S for Stack which is deallocating
      this stack: 0 except in dalostk%                               8A6J
      % ------- %
      (stackdecsize) CONSTANT=10; % entries in stack descriptor
      frame at bottom of each stack %                                8A6L
    % signal frame indices: BLT'd from globals indicated.  See their
```

```
   declarations for further information %
      (sgfrm) CONSTANT = 0; % signalling routine's fram (port id)%
                                                                    8A7A
         % SYSFRM %
      (sggco) CONSTANT = 1; %called catchphrase's co-return location
      %                                                             8A7B
         % SYSGCO %
      (sgsig) CONSTANT = 2; %signal value/name %                    8A7C
         % SYSSIG %
      (sggtp) CONSTANT = 3; % signal type %                         8A7D
         % SYSGTP %
      (sgg2) CONSTANT = 4; % signal parms %                         8A7E
         % SYSG2 %
      (sgg3) CONSTANT = 5; % signal parms %                         8A7F
         % SYSG3 %
      (sgg4) CONSTANT = 6; % signal parms %                         8A7G
         % SYSG4 %
      (sgcat) CONSTANT = 7;% catchphrase stack pointer for
      dispatched catchphrase %                                      8A7H
         % &SYSCAT %
      (sgbot) CONSTANT = 8; % "bottom" of stack for signal
      propogation%                                                  8A7I
         % SYSBOT %
      (sgstk) CONSTANT = 9; % stack pointer to base of stack
      propogating the signal %                                      8A7J
         % SYSSSTK %
      (sgnst) CONSTANT = 10; % nested scan flag %                   8A7K
         % NEXTSCAN %
      (sggn) CONSTANT = 11; % the signal number %                   8A7L
         % SYSGN %
      % ------- %
      (sigsiz) CONSTANT=12; % in signal in progress frame %         8A7N
   REF syscpt, syscat, syscpe, sysgpt, startup, recover;
   %.Pes;%
```

```
(sysrcv) % recover/startup/call/return code here %                    8B
   PROCEDURE % called for fatal error %
      (type, % error type code %
      why, % error code %
      loc); % associated location %
      (lsb) REF; % local stack bottom: used in looping over stacks %
                                                                      8B1D
      (lst) REF; % local stack top: used in looping over stacks %
                                                                      8B1E

   (bptrcv): % see arguments - label for debugging convenience %  8B2
      % check for recovery loop %
         IF NOT sysrip THEN % recovery in progress ? %
            BEGIN % first try - issue NOTE %
            sysrip _ TRUE;
            sysetc _ type; % error-type code %
            syswhy _ why;
            IF NOT linlink THEN
               BEGIN
               IF loc.RH IN [sysbtm.RH, (sysbtm.RH-sysbtm.LH).RH]
               THEN % a port %
                  loc _ [ loc.RH+1 ].RH; % get its last exit point %
               END
            ELSE % other stacks: is this a port in this stack? %
               BEGIN
               &lsb _ $gstack;
               DO % loop over all stacks %
                  BEGIN
                  &lst _ &lsb + lsb[stksize];
                  IF loc.RH IN [&lsb, &lst] THEN
                     BEGIN
                     loc _ [loc.RH + 1].RH;  % coreturn location %
                     EXIT LOOP;
                     END;
                  END
               WHILE (&lsb _ (lsb[stklink]).RH);
               END;
            sysloc _ loc;
            NOTE(unWind);
            END;
      % store error parameters and start over %
         !JSP R1,l10set; % reset stacks %
         syssys(TRUE); % recover %
      %.Pes;%
```

```
(l10start): % initial startup here %                                    8B3
   !JSP R1,l10set; % reset stacks %
   sysrip _ FALSE;
   syssys(FALSE); % recover %

(l10set): % initialize stack pointers (local label) %                   8B4
   %general call stack%
      S _ -gstksz; !HRL S,S; !HRRI S,gstack;
      M _ S;
      sysbtm _ S;
      gstack _ $sysufl;
      sysse _ $gstack+gstksz;
      % indicate only one stack thusfar %
         linlink _ 0;
      % Set up gstack stack descriptor %
         gstack[ stkmsave ] _
         gstack[ stkssave ] _
         gstack[ stklink ] _
         gstack[ stkblink ] _
         gstack[ stkzone ] _
         gstack[ stkdalo ] _
         gstack[ stkmdalo ] _
         gstack[ stksdalo ] _
         gstack[ stksig ] _ 0;
         gstack[ stksize ] _ gstksz;
   %pattern stack%
      p _ -pstksz; !HRL p,p; !HRRI p,pstack;
      swork _ endfil; % initialize for patterns %
   % spec stack %
      spsk _ spsk1;
   % reset clist change stack %
      RESET clchng; % before any string constructions are done%
      clmpty _ clchng;
   % return %
      !JRST 0(R1);
   %.Pes;%
```

```
   % returns, get here by JRST produced by compiler %
    (sysrtt): % return true %                                          8B5A
       !MOVEI mreg,1;

    (sysrtn): % return, value in mreg %                                8B5B
       !MOVE R4,S; % save old S for syspop - just in case %
       !MOVE S,M;
       !POP S,M;
       (systz1): !POPJ S,0;                                            8B5B4
       %+TRACE%                                                        8B5B5
          !MOVEM R1,systr1; !jobtm(); % get clock %
          !SUBM R1,systck; !EXCH R1,systck; % elapsed time %
          !SKIPGE R1; !MOVEI R1,0; % check wraparound %
          !HRLI R1,0; !JSR systwd; % write 0,,time: means return %
          !MOVE R1,systr1; !POPJ S,0; % return %
          %+TRACE%                                                     8B5B5F

    (sysrtf): % return false %                                         8B5C
       !MOVEI mreg,0;
       !MOVE R4,S; % save old S for syspop %
       !MOVE S,M;
       !POP S,M;
       (systz2): !POPJ S,0;                                            8B5C5
       %+TRACE%                                                        8B5C6
          !JRST systz1; % for tracing %
          %+TRACE%                                                     8B5C6B
    %.Pes;%
```

```
    % procedure entry, pcall. get here by JSP A4 produced by compier
    %
       (sysent): % procedure entry point (trace point) %              8B6A
          !PUSH S,M; % push old mark %
          !MOVE M,S; % set up new one %
          (systz3): !AOBJN S,0(A4); % return to procedure %           8B6A3
             % AOB puts blank on stack (coroutine return) %
             % also check for overflow (positive) %

       (sysovr): % stack overflow (follows sysent) %                  8B6B
          S _ sysbtm; M _ S;
          sysrcv(stkoverflow,$"general stack overflow",0);

       (syscent): % port entry (trace point) %                        8B6C
          !PUSH S,M; % push old mark %
          !MOVE M,S; % get new one %
          !PUSH S,A4; % setup co-return loc for ddt %
          (systz4): !JRST 0(A4); % return to port entry %             8B6C4
          %+TRACE% % make it look like pcall %                        8B6C5
             !MOVEM R1,systr1;
             !MOVE R1,systmd; !TRNN R1,1; !JRST notrce1; %check mode
             %
             !jobtm(); % get clock %
             !SUBM R1,systck; !EXCH R1,systck; % elapsed time %
             !SKIPGE R1; !MOVEI R1,0; % check wraparound %
             !HRLI R1,1; !JSR systwd; % write 1,,time: means pcall %
             !HRLI R1,0(A4); !HRR R1,1(M); !JSR systwd;
             %source,,dest%
             !MOVEI R1,0; !JSR systwd; % a zero for now %
             (notrce1):                                              8B6C5I
             !MOVE R1,systr1; !JRST 0(A4); % return to port entry %
             %+TRACE%                                                8B6C5K
          %.Pes;%
```

```
(pcall): % port-call code (trace point) %                    8B6D
   % mreg has port id to call; A4 contains return address; M
   has calling port id %
   !MOVEM A1,tempalsave;        % save A1 (argument/result) %
   !MOVEM mreg,tempmregsave;    % save mreg (destination port)
   %
   !HRRZS mreg;
   !MOVE A1,sysbtm;
   !CAILE mreg,0(A1);  % new port in current stack? %
   !CAILE mreg,0(S);   % It is if sysbtm < mreg < S %
      !JRST syspc2;    % New port not in current stack %
   !MOVEM A1,pcstack;  % set up previous stack pointer %
   (syspc1):           % pulse coroutine %                    8B6D9
      !SETZM sysfctn;  % reset system indicator %
      !MOVEM A4,1(M);  % save return in old frame %
      !MOVE A1,tempalsave;     % restore A1 %
      !MOVE mreg,tempmregsave; % restore full port %
      !EXCH mreg,M;    % setup new frame, old is argument %
   (systr5):                                                 8B6D10
      !JRST @1(M);     % call new one %
   %+TRACE%                                                  8B6D11
      !MOVEM R1,systr1;
      !MOVE R1,systmd; !TRNN R1,1; !JRST notrce; % check mode
      %
      !jobtm(); % get clock %
      !SUBM R1,systck; !EXCH R1,systck; % elapsed time %
      !SKIPGE R1; !MOVEI R1,0; % check wraparound %
      !HRLI R1,1; !JSR systwd; % write 1,,time: means pcall %
      !HRLI R1,0(A4); !HRR R1,1(M); !JSR systwd;
      %source,,dest%
      !MOVEI R1,0; !JSR systwd; % a zero for now %
      (notrce):                                              8B6D11I
      !MOVE R1,systr1; !JRST @1(M); % call port %
      %+TRACE%                                               8B6D11K
   (syspc2): % New port not in this stack.  Loop through all
   stacks %                                                  8B6D12
      % save state (M, S, signal status) of current stack %
         !MOVEM A1,pcstack;    % save calling stack ID %
         !MOVEM S,stkssave(A1);        % Save current S %
         !MOVEM M,stkmsave(A1);        % save M %
         % save sig state, if any in progress, if this is not
         a continue PCALL.  (sigsave checks to see if there is
         any signal in progress.)  SYSFCTN is TRUE if the
         PCALL comes from continue (i.e., HELP, NOTE, ABORT,
         or CONTINUE) or RESUME, FALSE otherwise. %
            !SKIPE sysfctn;
               !JRST syspc3;
            % save registers around sigsave %
               !MOVEM R15,srg15;
               !MOVEI R15,srgsav;
               !BLT R15,srg14;
               !MOVE R15,srg15;
            sigsave( A4);
            % restore registers %
               !HRLZI R15,srgsav;
               !BLT R15,R15;
```

```
        (syspc3):       % point to first stack %          8B6D12B
           !SKIPN A1,linlink;    % more than 1 stack? %
              !JRST pclerr;      % no - give error %
        (pcloop):                                        8B6D12C
           !EXCH S,stkssave(A1); % get old S %
           !CAILE mreg,0(A1);    % in this stack? %
           !CAILE mreg,0(S);
              !JRST pclpend;     % no. look at next stack. %
           !MOVEM S,stkssave(A1);        % restore saved S %
           % We have found the proper stack for the new port %
              !MOVEM A1,sysbtm;  % set up sysbtm to new stack %
              % restore signal state for new stack if this is
              not a continue pcall %
                 !SKIPE sysfctn;
                    !JRST syspc4;
                 % save registers around sigrestore %
                    !MOVEM R15, srg15;
                    !MOVEI R15, srgsav;
                    !BLT R15, srg14;
                    !MOVE R15, srg15;
                 sigrestore();
                 % restore registers %
                    !HRLZI R15, srgsav;
                    !BLT R15, R15;
              (syspc4):                                  8B6D12C5C
                 !JRST syspc1;          % dispatch PCALL %
        (pclpend):                                       8B6D12D
           !EXCH S,stkssave(A1);
           !SKIPE A1,stklink(A1);        % any more stacks? %
              !JRST pcloop;      % Yes. %
        (pclerr):                                        8B6D12E
           !MOVE A1,tempa1save;  % No. Restore A1 %
           !MOVE mreg,tempmregsave;      %. Restore mreg %
           !JRST sysnxp;         % error %
     %.Pes;%
```

```
    % fatal error points, JSP,A4 produced by compiler %

       (sysnxp): % non-existant port called %                          8B7B
          sysrcv(programbug,$"non-existant port called",A4);

       (syscer): % coroutine called as procedure %                     8B7C
          sysrcv(programbug,$"coroutine called as procedure",A4);

       (syscte): % catchphrase termination error %                     8B7D
          sysrcv(programbug,$"no catchphrase termination
          specified",A4);

       (syster): % catchphrase fall-thru error %                       8B7E
          sysrcv(programbug,$"catchphrase did not dispatch
          signal",A4);

       (sysufl): % stack underflow (strange) %                         8B7F
          sysrcv(programbug,$"general stack underflow",0);

    END.  % of sysrcv procedure above %
    %.Pes;%
```

```
(syssys) % initialize system coroutines, etc %                          8C
   PROCEDURE % this must be done from a procedure %
      (status); % TRUE if recovering %                                  8C1A
   % LOCAL  for catchphrase %
      (tcatp) REF; % temporary pointer to catchframe stack %            8C2A
      (mask); % used by catchphrase to contain a mask which will
      cause  the bit corresponding to the signal which has been in
      progress to be shut off. %                                        8C2B
   % initialize catchphrase stack %
      &syscpe _ &syscat _ $syscstk;
      &sysgpt _ $sysgstk;
      sysgn _ tsysgn _ sysbfg _ 0;
      syshcon _ FALSE;
   % open system coroutines %
      OPENPORT syshelp( helptype :[syshlp]); % HELP %
      OPENPORT sysctr(:[syscon]); % CONTINUE %
      OPENPORT syshelp( aborttype :[sysabt]); % ABORT %
      OPENPORT syshelp( notetype :[sysnot]); % NOTE %
      OPENPORT sysrsume(:[sysres]); % RESUME %
      OPENPORT sysvoke(:[sysinv]); % INVOKE %
   % invoke catch-all catchphrase %
      INVOKE(syscatch);
      sysctchall _ &syscpe-catsiz; % save system catch pointer %
   % give control to program %
      sysrip _ FALSE;
      IF status THEN recover(sysetc,syswhy,sysloc)
      ELSE startup();
         % recover and startup are the addresses of appropriate
         procedures specified for the L10 program.  They<LF> should
         never return. %
      sysrcv(programbug,$"startup/recover procedure returned",0); %
      don't return %

   (syscatch) CATCHPHRASE;  % TOP CATCHPHRASE in L10 ENVIRONMENT %
                                                                        8C7
      BEGIN
      % unmark catchphrase stack for the signal which was in
      progress %
         mask _ bits[sysgn] .X -1;
         FOR &tcatp _ $syscstk UP catsiz UNTIL >= &syscpe DO
            IF tcatp[catstk] = syssstk THEN
               tcatp[catmrk] _ tcatp[catmrk] .A mask;
      CASE SIGNALTYPE OF
         =notetype: % this is where NOTEs are resumed %
            BEGIN
            sysgtp _ helptype; % smash type !! %
            RESUME;
            END;
         =helptype: RESUME(nohelp);
         ENDCASE
            BEGIN
            % params in global safe cells %
               syszgn _ SIGNAL; syszgt _ SIGNALTYPE;
               syszg2 _ sysg2; syszg3 _ sysg3; syszg4 _ sysg4;
            sysrcv(uncaughtabort,$"ABORT to runtime code",sysfrm);
            END;
```

```
    END;
END.
%.Pes;%
```

```
% L10 Runtime TRACE FACILITY %
   (systrc) % system trace invoke routine %                          8D1
      PROCEDURE(
         mempg, % memory page number to use for trace %
         mode, % trace mode specifier %
         typeout); % zero or a jfn to write start/fin messages on %
      % call/return etc. trace setup. trace code currently not
      protected from traced events inside PSI's! %
      %+TRACE%                                                       8D1C
         IF systen=0 OR systfg THEN RETURN; % no-op if already on %
         % setup globals %
            systpn _ mempg;
            systmd _ mode;
            systpc _ 0; % output file page count %
            systtp _ typeout;
            systpt _ systpn*512; % AOBJ pointer into page %
            systpt.LH _ -512; % count %
         % open file %
            IF NOT SKIP !gtjfn(600001B6, $"TRACE.DATA"+chbmty) THEN
               BEGIN
               IF systtp THEN !sout(systtp, $"gtjfn failed
               -trace"+chbmty, -19);
               RETURN;
               END;
            systjfn.LH _ R1;
            IF NOT SKIP !openf(R1, 2200001B5) THEN
               BEGIN
               IF systtp THEN !sout(systtp, $"openf failed
               -trace"+chbmty, -19);
               RETURN;
               END;
            !chfdb(11B6+R1, 77B8, 22B8); % byte size = 18 %
         % put out message %
            IF systtp THEN
               BEGIN
               !sout(systtp, $"Tracing "+chbmty, -8);
               !bout(systtp, EOL);
               END;
         % replace secret cells %
            systi1 _ systz1 := systi1; % replace secret cells %
            systi2 _ systz2 := systi2;
            systi3 _ systz3 := systi3;
            systi4 _ systz4 := systi4;
            systi5 _ systz5 := systi5;
         % go... %
            systfg _ TRUE;
            systck _ !jobtm();
         %+TRACE%                                                    8D1C7
      RETURN;
      %+TRACE%                                                      8D1E
         (systj3);                                                  8D1E1
            !MOVEM R1,systr1; !jobtm(); % get clock %
            !SUBM R1,systck; !EXCH R1,systck; % elapsed time %
            !SKIPGE R1; !MOVEI R1,0; % check wraparound %
            !HRLI R1,-1(A4); !JSR systwd; % write adr,,time: call %
            !MOVE R1,systr1; !AOBJN S,0(A4); !JRST sysovr; % go %
```

```
      (systi1): !JFCL;                                                  8D1E2
      (systi2): !JFCL;                                                  8D1E3
      (systi3): !JRST systj3;                                           8D1E4
      (systi4): !JFCL;                                                  8D1E5
      (systi5): !JFCL;                                                  8D1E6
      (systwd): % JSR routine to write word into trace page %
                                                                        8D1E7
         !JFCL; % the return location stored here %
         !EXCH R3,systpt; % get pointer/save R3 %
         !MOVEM R1,0(R3); % store word %
         !AOBJN R3,nomap; % count %
         !MOVSI R1,485; % end of page, pmap it %
         !HRR R1,systpn; % core page number %
         !EXCH R2,systjfn; % get jfn in LH, save R2 %
         !HRR R2,systpc; % file page number %
         !MOVSI R3,140400B; % write access, copy on write %
         !pmap();
         !EXCH R2,systjfn; % restore R2 %
         !AOS systpc; % up page count %
         !HRRZ R3,systpn; % initialize pointer again %
         !LSH R3,9; % make page number an address %
         !HRLI R3,-1000B; % initialize count %
         !jobtm(); !MOVEM R1,systck; % make pmap invisible %
         (nomap):                                                       8D1E7Q
            !EXCH R3,systpt; % restore pointer %
            !JRST @systwd; % return %
      %+TRACE%                                                          8D1E8
   END.
   %.Pes;%
```

```
(systoff) % system trace - turn it off %                                8D2
   PROCEDURE; % no arguments %
   %+TRACE%                                                             8D2B
      IF systm=0 OR systfg=0 THEN RETURN; % no-op if already off
      %
      % set EOF byte count for file %
        R3 _ systpc; !LSH R3,10; % byte count already pmapped %
        R4 _ systpn; !LSH R4,9; % page address %
        !HRRZ R2,systpt; !SUBI R2,0(R4); % words in this page %
        !LSH R2,1; % times 2 for bytes %
        !ADDI R3,0(R2); % total bytes in file %
        !chfdb(12B6+systjfn.LH, -1, R3);
      % map out last page %
        systpt.LH _ 0; % force it % !JSR systwd;
        R4 _ systpn; !LSH R4,9; !MOVEM R4,0(R4); % make it
        private %
        IF NOT SKIP !closf(systjfn.LH) THEN
           IF systtp THEN !sout(systtp, $"closf failed
           -trace"+chbmty, -19);
      % restore secret cells %
        systi1 _ systz1 := systi1;
        systi2 _ systz2 := systi2;
        systi3 _ systz3 := systi3;
        systi4 _ systz4 := systi4;
        systi5 _ systz5 := systi5;
      % done %
        systfg _ FALSE;
        IF systtp THEN
           BEGIN
           !sout(systtp, $"Trace off"+chbmty, -9);
           !bout(systtp, EOL);
           END;
      %+TRACE%                                                          8D2B6
   RETURN;
   END.
   %.Pes;%
```

```
% Signal and Catchphrase Coroutines and Procedures %
   (syshelp)  % ABORT/NOTE/HELP coroutine %                              8E1
      COROUTINE ( type);     % signal type %
      LOCAL
         sigval, % temp slot for signal value %
         s2, s3, s4; % other args %
      PORT ENTRY EXIT sigval _ PCALL (:s2, s3, s4);
      LOOP
         BEGIN
         % save global signal info (only needed if we didn't switch
         stacks to get here) %
            IF pcstack = sysbtm THEN sigsave( PORT );
         % set up global info to reflect signal being generated now
         %
            syssig _ sigval; % set SIGNAL %
            sysg2 _ s2;      % global args %
            sysg3 _ s3;
            sysg4 _ s4;
            sysfrm _ PORT;   % signalling port id %
            sysgtp _ type;   % set signal type %
            syssstk _ pcstack;       % signalling stack id %
            sysinfo _ 0; % the global signal information is not on
            the signal stack %
         % keep track of number of signals in progress %
            BUMP sysgn;      % for generating stack %
            BUMP tsysgn;     % for entire system %
         % search for proper catachphrase from top of catchphrase
         stack %
            &syscat _ &syscpe;
         % indicate for sysctn that we came from here and not from a
         CONTINUE in a catchphrase %
            syshcon _ TRUE;
         % set "bottom" for this signal (to generating frame for
         NOTEs Unwind or Return).  SYSBFG is set to a non-zero value
         in SYSPOP (=> RETURN with catchphrases to be dropped) and
         SYSTRM (=> UNWIND).  This value delimits the range of the
         frames on the call stack which will see these signals.
         Otherwise, SYSBFG is FALSE:  we therefore set SYSBOT to the
         bottom of the signalling stack.  Signals will propogate
         only in the stack in which they are generated! %
            IF sysbfg THEN sysbot _ sysbfg := 0
            ELSE sysbot _ syssstk;
         % now call CONTINUE %
            sigval _ PCALL [syscon] (:s2, s3, s4);
         END;
      END.
      %.Pes;%
```

```
(sysctn) % CONTINUE coroutine %                                    8E2
   % This coroutine searches the catch stack from syscat to
   bottom and calls the first eligible catchphrase it finds.
   syscat setup by syshelp, sysabort or sysnte - or a previous
   CONTINUE. Starts with next catchphrase DOWN from syscat. %
   COROUTINE;
   LOCAL port, nur, tcatp REF;
   PORT ENTRY EXIT PCALL(: [port] );
      % Can't reference PORT.RH %
   LOOP
      BEGIN
      % if from a catchphrase, must get proper signal info from
      sig stack into global cells (info will already be in global
      cells if we didn't have to switch stacks to get here) and
      must restore coroutine location in stack frame ( [PORT+1]
      is co-loc on PDP-10 ): syshcon  is TRUE if NOTE, ABORT, or
      HELP; FALSE if CONTINUE, i.e. from a catchphrase %
         IF syshcon THEN syshcon _ FALSE
         ELSE
            BEGIN
            IF pcstack # sysbtm THEN
               % we switched stacks to get here; thus we must put
               signal info from sig stack into global area %
               BEGIN
               savebtm _ sysbtm := pcstack;
               sigrestore();
               sysbtm _ savebtm;
               END;
            [port.RH+1] _ sysgco;
            END;
      % If Note (Unwind or Return), we will propogate the signal
      only to those frames more recently placed in the flow of
      control then the frame generating the signal (i.e., its
      subframes) %
         nur _ (IF sysbot=sysbtm THEN FALSE ELSE TRUE);
      LOOP % over catchphrase stack: find eligible cachphrase %
         BEGIN
         % get to next catchphrase in catstack.  If at bottom,
         set to generate SYSCATCHALL; if not, decide if it should
         be dispatched. %
         IF (&syscat _ &syscat - catsiz) >= $syscstk THEN
            BEGIN % not at bottom: should this be dispatched? %
            % only look at catchphrases that belong to the same
            stack that generated the signal %
               IF syscat[catstk] # syssstk THEN REPEAT LOOP;
            IF NOT nestscan AND syscat[catmrk] THEN
               BEGIN
               % we are not yet in the middle of a nested signal
               scan, but since this catchphrase has been seen by
               some signal (since syscat[catmrk] is non-zero), we
               start a scan from the bottom of the stack and go
               up the stack until we find the first catchphrase
               seen by the immediately preceding signal: that one
               is a candidate for seeing the current signal.
               This catchphrase defines where a branch occurred
               in the thread of control. %
```

```
            nestscan _ TRUE;
            FOR &tcatp _ $syscstk UP catsiz UNTIL >= &syscat
            DO
               IF tcatp[catmrk] .A bits[sysgn-1] THEN EXIT
               LOOP;
            &syscat _ &tcatp;
            END;
         % This catchphrase is marked as having seen this
         signal. %
            syscat[catmrk] _ syscat[catmrk] .V bits[sysgn];
         % if this catachphrase is disabled, or if this is a
         Note (Unwind or Return) and this catachphrase is out
         of range, then we must find another catchphrase,
         otherwise we exit this loop and dispatch this
         catchphrase %
            IF (syscat[catenc] <= 0) OR (nur AND
            (syscat[catown] < sysbot))
               THEN REPEAT LOOP % check another catchphrase %
               ELSE EXIT LOOP; % dispatch the catchphrase %
         END
      ELSE  % no more catchphrases on stack: dispatch the
      catchall catchphrase %
         BEGIN
         &syscat _ sysctchall; % system catchall - will resume
         NOTE %
         EXIT LOOP;
         END;
      END; % of catchphase scan loop %
   % save coreturn location, then change it to point to
   catchphrase so catchphrase is actually dispatched by
   PCALLing the owning frame %
      sysgco _ [syscat[catfrm]+1] := syscat[catloc];
   % setup catchphrase parameter %
      syscpm _ syscat[catprm];
   (bptctn): % syscat points to catch frame being activated %
                                                          8E2E7
   % call catchphrase %
      sysfctn _ TRUE;
         % checked in pcall to see whether the global signal
         stuff should be changed.  IT WILL NOT BE SINCE THIS
         PCALL IS A "SYSTEM" PCALL TO EXECUTE THE SIGNAL! %
      PCALL [syscat[catfrm]] (0,sysg2,sysg3,sysg4:[port]);
   END; % of coroutine body "loop" %
END.
%.Pes;%
```

```
(sysrsume) % RESUME coroutine %                                    8E3
   % This coroutine RESUMEs code when a catchphrase does a RESUME
   %
   COROUTINE;
   LOCAL
      port, % for port.RH references %
      f, % flag %
      res1, res2, res3, res4, % results to pass on %
      sport; % temp cell for port %
   PORT ENTRY EXIT res1 _ PCALL (: [port] res2, res3, res4);
   LOOP
      BEGIN
      % get signal info into global area (already there if we
      didn't switch stacks) %
         IF pcstack # sysbtm THEN
            BEGIN
            savebtm _ sysbtm := pcstack;
            sigrestore();
            sysbtm _ savebtm;
            END;
      % check signal status for screwups %
         IF SIGNALTYPE # helptype THEN
            sysrcv(programbug,$"attempt to resume a
            NOTE/ABORT",[port.RH+1]);
         IF sysgn <= 0 THEN % in progress count %
            sysrcv(programbug,$"attempt to RESUME with no signal
            in progress",[port.RH+1]);
      % restore co-loc in catchphrase frame %
         [port.RH+1] _ sysgco;
      % get current values before poping %
         sport _ sysfrm; % signal from this port %
      popsig();
      syscpm _ syscat[catprm]; % restore parameter %
      (bptres): % (N)+4 is res1, others follow %                   8E3E8
      sysfctn _ TRUE;
         % checked in pcall to see whether the global signal
         stuff should be changed.  IT WILL NOT BE SINCE THIS
         PCALL IS A "SYSTEM" PCALL TO EXECUTE THE SIGNAL! %
      res1 _ PCALL [sport] (res1, res2, res3, res4
         : [port] res2, res3, res4); % RESUME %
      END;
   END.
   %.Pes;%
```

```
(sysvoke) % INVOKE coroutine %                                          8E4
   % This coroutine invokes a catchphrase and sets termination
   location.  It also smashes return location for procedures.
   NOTE: if caller is a coroutine, must locate owning procedure
   in order to smash its return location.  Real return locations
   are kept in the catchphrase frame %
   COROUTINE;
   % PCALL arguments:
      cloc, catchphrase location
      tloc; TERMINATE in this catchphrase instance means GOTO
      here %
   LOCAL port; % temp store for port id %
   PORT ENTRY
      EXIT syscpe[catloc] _ PCALL(: syscpe[cattloc],
      syscpe[catprm]);
   LOOP
      BEGIN
      % initialize catchphrase frame %
         % catchphrase location stored in PCALL %
            syscpe[catfrm] _ PORT; % catchphrase port (frame) %
         % termination location stored in PCALL %
         syscpe[catenc] _ 1; % enable it %
         % save current stack pointer of stack invoking
         catchphrase %
            syscpe[cats] _
               IF pcstack = sysbtm THEN S
               ELSE [pcstack.RH][stkssave];
         % indicate it hasn't seen any signals yet %
            syscpe[catmrk] _ 0;
         % indicate which stack owns this catchphrase %
            syscpe[catstk] _ pcstack;
      (bptinv): % syscpe points to catch frame %                        8E4F3
      % find owning procedure and save/change return location %
         port _ PORT;
         WHILE (NOT sysprc(port)) AND ([port.RH] # $sysuf1) DO
            port _ sysbak(port);
         syscpe[catown] _ port;
         IF ([port.RH-1].RH # $syspop) AND ([port.RH] # $sysuf1)
         THEN % save/change return %
            syscpe[catrtn] _ [port.RH-1] := $syspop
         ELSE syscpe[catrtn] _ 0; % zero means already changed %
      % bump catchphrase frame pointer %
         IF ( &syscpe _ &syscpe + catsiz) > $sysce THEN
            sysrcv(programbug,$"catchphrase stack
            overflow",PORT);
      syscpe[catloc] _ PCALL (:syscpe[cattloc], syscpe[catprm]);
      END;
   %.Pes;%
```

```
(syspop): % label to drop catchphrases on RETURN %          8E4H
   % first "undo" the return so sig will work %
      !MOVE M,S; !ADD M,=2000002B; % M _ S+2 %
      !MOVE S,R4;
   % first save args and mreg %
      !PUSH S,A1; !PUSH S,mreg; %return values that will be
      smashed %
      !PUSH S,R1; !PUSH S,R2; !PUSH S,R3;
   % put out "return" NOTE %
      sysbfg _ M; % above and here only %
      NOTE(return);
      !POP S,R3; !POP S,R2; !POP S,R1; % restore those args
      now %
   &syscpt _ &syscpe;
   systloc _ 0; % this will be return loc %
   WHILE (&syscpt _ &syscpt - catsiz) >= $syscstk DO
      IF syscpt[catfrm] IN [M, S) THEN
         % NOTE:
            Old mark was S+2. Two PUSH's make it =S. Want to
            remove all his catchphrases and those of all
            routines above him in stack. There may be phrases
            for routines below him in stack, above his phrases
            in catch stack. %
         BEGIN
         IF syscpt[catrtn] AND syscpt[catown] = M THEN
            systloc _ syscpt[catrtn];
         % mark the catchphrase to be dropped %
            drpcat _ syscpt[catenc] _ dropped;
         END;
   IF NOT tsysgn AND drpcat THEN sysdpall();
   IF systloc THEN
      BEGIN
      [M.RH-1] _ systloc; % resore return loc %
      !POP S,mreg;
      !POP S,A1; % restore results %
      (bptrtn): % break just before real return %          8E4H8E
      % just like sysrtn, not traced %
         !MOVE S,M;
         !POP S,M;
         !POPJ S,0; % go %
      END;
   sysrcv(programbug,$"syspop: cannot find return for
   procedure",S.RH);
   % means did not find return for this procedure - major
   screwup %
END.
%.Pes;%
```

```
(systrm) % terminate this signal (and return to termloc) %      8E5
   % This procedure performs TERMINATE command.  It terminates
   signals and gives control to the TERMINATing routine at the
   location specified when the catchphrase was invoked (i.e.
   termloc ) %
   PROCEDURE;
   LOCAL
      tloc, % to save termination loc - global no good here %
      savs, savm, trmown; % temps to save S, M, TERM owner's port
       %
   tloc _ systloc _ syscat[cattloc]; % save the termloc in global
   %
   trmown _ syscat[catown]; % position in flow of control for
   TERMINATOR %
   (bpttrm): % here, systloc is term. location %                8E5F
   IF sysgn <= 0 THEN
      sysrcv(programbug,$"TERMINATE with no sig in
      progress",[M.RH-1]);
   IF sysbtm # syssstk THEN
      sysrcv(programbug,$"TERMINATE for signal in another
      stack",[M.RH-1]);
   savs _ syscat[cats];
   savm _ syscat[catfrm];
   IF SIGNALTYPE = notetype THEN
      sysrcv(programbug,$"attempt to TERMINATE a NOTE",[M.RH-1]);
   popsig(); % pop 1 signal %
   WHILE sysgn>0 AND syscat[catown] >= trmown DO popsig();
      % remove all signals generated by routines lower in flow of
      control than the routine that is TERMINATING %
   sysbig _ savm+1; % set lower limit for NOTE (NOT this frame) %
   NOTE(unwind);
   % now remove all catchphrases for lower routines %
      &syscpt _ &syscpe;
      WHILE (&syscpt _ &syscpt - catsiz) >= $syscstk DO
         IF syscpt[catfrm] > savm THEN
            % mark the catchphrase to be dropped %
               drpcat _ syscpt[catenc] _ dropped;
      IF NOT tsysgn AND drpcat THEN sysdpall();
   systloc _ tloc; % restore that global. may have been used %
   S _ savs; % restore stack %
   M _ savm;
   GOTO [systloc]; % GOTO termination location %
   END.
   %.Pes;%
```

```
(sysena) % enable catchphrase %                                    8E6
   PROCEDURE
      (catchnam); % address of catchphrase %                       8E6A1
   sysabl(catchnam, [M] % caller's mark %, 1);
   RETURN;
   END.
   %.Pes;%
```

```
(sysdis) % disable catchphrase %                                    8E7
   PROCEDURE
      (catchnam); % address of catchphrase %                        8E7A1
   sysabl(catchnam, [M] % caller's mark %, -1);
   RETURN;
   END.
   %.Pes;%
```

```
% Signal and Catchphrase Utility Procedures %
   (sysabl) % manipulate catchphrase enable count %                    8F1
      PROCEDURE
         (catchname, % address of catchphrase %
         oldmark, % mark for caller %
         n); % count %
      LOCAL
         catp REF; % catchphrase stack pointer %
      &catp _ &syscpe;
      % now search for most recent instance of catchname %
         WHILE (&catp _ &catp - catsiz) > $syscstk DO
            IF % right one %
               catp[catloc] = catchname AND
               catp[catfrm] = oldmark AND
               catp[catenc] # dropped THEN
                  BEGIN
                  catp[catenc] _ catp[catenc] + n;
                  EXIT;
                  END;
         % no-op if not found %
      RETURN;
      END.
      %.Pes;%
```

```
(sysenc) % return enable count given catchphrase adr %                  8F2
   PROCEDURE
      (catchname); % address of catchphrase %                           8F2A1
   LOCAL
      catp REF, % catchphrase stack pointer %
      oldmark; % caller's mark %
   oldmark _ [M];
   &catp _ &syscpe;
   % now search for instance of catchphrase %
      WHILE (&catp _ &catp - catsiz) > $syscstk DO
         IF catp[catloc] = catchname AND catp[catfrm] = oldmark
         THEN
            RETURN(catp[catenc]);
      % zero if not found %
   RETURN(0);
   END.
   %.Pes;%
```

```
(sysdrp) % drop catchphrase %                                    8F3
   PROCEDURE
      (catchname); % address of catchphrase %                    8F3A1
   % drops a catchphrase given address, or drops all catchphrases
   belonging to calling routine if argument is -1 %
   LOCAL
      prevcat REF, % pointer to previous catchphrase for caller %
      catp REF, % catchphrase stack pointer %
      oldmark; % caller's mark %
   oldmark _ [M];
   &catp _ &syscpe;
   &prevcat _ 0;
   % now search for instance of catchphrase and delete it %
      WHILE (&catp _ &catp - catsiz) > $syscstk DO
         IF catp[catfrm] = oldmark AND (catp[catenc] # dropped)
         AND
            ( catp[catloc] = catchname OR catchname=-1 ) THEN
               BEGIN
               % restore return location or move it to previous
               catchphrase %
               IF catp[catrtn]#0 THEN % return loc in it %
                  BEGIN
                  IF &prevcat THEN prevcat[catrtn] _ catp[catrtn]
                  ELSE [catp[catown].RH-1] _ catp[catrtn];
                  % mark the catchphrase to be dropped %
                     drpcat _ catp[catenc] _ dropped;
                  EXIT; % we know we are done here %
                  END
               ELSE
                  % mark the catchphrase to be dropped %
                     drpcat _ catp[catenc] _ dropped;
               IF catchname#-1 THEN EXIT;
               END
         ELSE IF catp[catown] = oldmark THEN &prevcat _ &catp;
            % save prev cat pointer %
         % ignore if does not belong to oldmark %
      IF NOT tsysgn AND drpcat THEN sysdpall();
      % no-op if not found %
   RETURN;
   END.
   %.Pes;%
```

```
(sysdpall) % drop all marked catchphrases %                          8F4
   PROCEDURE;
   % to be used by runtime routines only; reset drpcat falg to
   FALSE %
   (catp) REF; % temporary pointer for looping over catchphrase
   stack %                                                           8F4C
   &catp _ &syscpe;
   WHILE (&catp _ &catp - catsiz) > $syscstk DO
      BEGIN
      IF catp[catencl=dropped THEN
         sysdp1(&catp);
      END;
   drpcat _ FALSE;
   RETURN;
   END.
   %.Pes;%
```

```
(sysdp1) % drop one catchphrase given catch stack pointer %    8F5
   PROCEDURE
      (catp REF); % catchphrase sack pointer of one to drop %
   % to be used by runtime routines only %
   IF &catp = &syscpe - catsiz THEN &syscpe_&catp % on top %
   ELSE
      BEGIN
      &syscpe _ &syscpe - catsiz; % step down one %
      % want to move from &catp+catsiz to &catp till we store in
      &syscpe-catsiz %
      R4.LH _ &catp+catsiz;
      R4.RH _ &catp;
      A1 _ &syscpe+catsiz;
      !BLT R4,@A1;
      END;
   RETURN;
   END.
   %.Pes;%
```

```
% Signal and Catchphrase State Saving Primitives %
   (sigsave) PROCEDURE  % save global signal state in signal stack %
                                                                    8G1
      % FORMALS %
         (rloc);      % return location for error messages %      8G1A1
      % does nothing if no signal in progress for previous stack.
      Uses syssstk (signalling stack).  Thus, if sysgn is non-zero
      syssstk MUST be set up.  This is done in syshelp.  It is reset
      in sigrestore.  %
      % DECLARATIONS %
         (tsigptr) REF;                                             8G1C1
         (endadr);                                                  8G1C2
      IF sysgn THEN
         BEGIN
         [syssstk.RH][stksig] _ sysgn;
         FOR &tsigptr _ $sysgstk UP sigsiz UNTIL >= &sysgpt DO
            IF (tsigptr[sggn] = sysgn) AND (tsigptr[sgstk] =
            syssstk) THEN
               EXIT LOOP;
         IF &tsigptr + sigsiz > $sysge THEN
            sysrcv(stkoverflow,$"signal stack overflow",rloc);
         A1.LH _ $sysfrm;   A1.RH _ sysinfo _ &tsigptr;
         endadr _ &tsigptr + sigsiz - 1;
         !BLT A1,@endadr;
         IF &tsigptr = &sysgpt THEN &sysgpt _ &sysgpt + sigsiz;
            % We have added a new signal to the stack %
         END;
      RETURN;
      END.

   (sigrestore) PROCEDURE;  % restore global signal state from sig
   stack %                                                          8G2
      % NOTE: *** CAN NOT USE ANY REGISTERS EXCEPT A1 *** %
      % Uses sysbtm, absolute bottom of the cutrent stack %
      % DECLARATIONS %
         (tsigptr) REF;                                             8G2C1
         (endadr);                                                  8G2C2
         (signo);                                                   8G2C3
      IF signo _ [sysbtm.RH][stksig] THEN
         BEGIN
         FOR &tsigptr _ $sysgstk UP sigsiz UNTIL >= &sysgpt DO
            BEGIN
            IF (tsigptr[sggn] = signo) AND (tsigptr[sgstk] = sysbtm)
            THEN
               EXIT LOOP;
            END;
         END
      ELSE &tsigptr _ &sysgpt;
      IF &tsigptr = &sysgpt THEN
         BEGIN
         sysinfo _ syssstk _ sysfrm _ 0;
         R1.LH _ $sysfrm;   R1.RH _ $sysfrm + 1;
         END
      ELSE
         BEGIN
         R1.LH _ sysinfo _ &tsigptr;   R1.RH _ $sysfrm;
```

```
        syssstk _ sysbtm;
        END;
    endadr _ $sysfrm + sigsiz - 1;
    !BLT R1,@endadr;
    RETURN;
    END.

(popsig) % pop signal-in-progress frame from sysgstk: called from
systrm and sysresume %                                              8G3
    % This routine reverses pushsig operation - recover on
    underflow %
    PROCEDURE;
    (temp);                                                         8G3C
    (svsybtm);                                                      8G3D
    (tsigptr) REF;                                                  8G3E
    (endadr);                                                       8G3F
    % unmark all catchphrases that saw this signal %
        temp _ bits[sysgn] .X -1;
        FOR &tsigptr _ $syscstk UP catsiz UNTIL >= &syscpe DO
            IF tsigptr[catstk] = syssstk THEN
                tsigptr[catmrk] _ tsigptr[catmrk] .A temp;
    IF sysinfo % the signal is on the stack: remove it % THEN
        BEGIN
        % check for underflow %
            IF (&sysgpt-sigsiz) < $sysgstk THEN
                sysrcv(programbug,$"signal stack underflow",0);
        % find this signal on sig stack and collapse the sig stack
        %
            IF sysinfo = &sysgpt - sigsiz THEN
                &sysgpt _ sysinfo
            ELSE
                BEGIN
                &sysgpt _ &sysgpt - sigsiz;
                R4.LH _ sysinfo + sigsiz;
                R4.RH _ sysinfo;
                endadr _ &sysgpt;
                !BLT R4,@endadr;
                % sysinfo will be set in sigrestore; should be set by
                here to reflect true state of the world %
                END;
        END;
    % restore prior signal %
        svsybtm _ sysbtm := syssstk;
    % move from sig stack to globals %
        sigrestore();
        % sysgn was restored by sigrestore %
        IF [sysbtm.RH][stksig] THEN BUMP DOWN [sysbtm.RH][stksig];
        IF tsysgn THEN BUMP DOWN tsysgn;
        sysbtm _ svsybtm;
    RETURN;
    END.
    %.Pes;%
```

```
% Signal and Catchphrase State Query Primitives %
   (sysprc) % is given stack frame for a PROCEDURE? %                    8H1
      % Return TRUE if port id P (frame pointer) is that of a
      procedure, false if that of a coroutine. Method: look at
      return location - if coroutine, it contains port id of caller
      which points into stack rather than to code (and, of course,
      there is no code on the stack.  We look first at stack pointed
      to by sysbtm, then go on through all other stacks if
      necessary. %
      PROCEDURE
         (port); % the port id %                                        8H1B1
      (pid);                                                           8H1C
      (teststk);                                                       8H1D
      IF (pid _ [port.RH-1].RH) IN [sysbtm.RH,
      (sysbtm.RH-sysbtm.LH).RH] THEN
         RETURN (FALSE);
      IF NOT (teststk _ linlink) THEN RETURN(TRUE);
      LOOP % over all stacks from the beginning: must end when
      stklink field 0 at end of chain %
         BEGIN
         IF pid IN [teststk.RH, (teststk.RH-teststk.LH).RH] THEN
            RETURN (FALSE);
         IF NOT (teststk _ [teststk.RH][stklink]) THEN RETURN(TRUE);
         END;
      END.

   (sysbak) % given stack frame,, back up one frame %                   8H2
      % Return frame pointer (port id) P for the frame that is
      "back" one frame in stack (back=toward bottom) %
      PROCEDURE
         (port); % the port id = frame pointer %                       8H2B1
      RETURN ( [port.RH] ); % gets previous mark %
      END.

   (sysclr) % return a procedure's caller's address %                   8H3
      % sysclr() inside a procedure will give you your return
      address %
      PROCEDURE;
      RETURN( [ [M.RH].RH-1 ] );
      END.
      %.Pes;%
```

```
(errmsg) % error message writing procedure %                              8I
   PROCEDURE
      (jfn, % output jfn %
      why REF, % reason %
      loc); % associated location %
   % put out CRLF %
      !bout(jfn,EOL);
      !sout(jfn,$"L10 RUNTIME ERROR: "+chbmty,0);
   % find reason %
      IF why.L > why.M OR why.L NOT IN [1,200] THEN
         &why _ $"bad error string passed to ERRMSG";
      !sout(jfn, &why+chbmty,0);
   % put out location %
      !bout(jfn,EOL);
      !sout(jfn,$"LOCATION: "+chbmty,0);
      CASE loc OF
         =0: !sout(jfn,$"unknown location (zero given)"+chbmty,0);
         ENDCASE
            BEGIN
            IF NOT SKIP !nout(jfn,loc,8) THEN !JFCL;
            !sout(jfn,$" octal"+chbmty,0);
            END;
      !bout(jfn,EOL);
   RETURN;
   END.
   %.Pes;%
```

```
% Stack allocation/deallocation %
   (alostk)   % alocate & openport coroutine stack %
   PROCEDURE (astk REF,nstksize, zone, acoroutine REF, nargs %, +
   nargs arguments for the coroutine % % => [port] coroutine's
   results %);                                                    8J1
      % Procedure description
         FUNCTION
            A call on this procedure is logically equivalent to
            allocating a new stack and then OPENPORTing a coroutine
            that lives at the base of the new stack
         ARGUMENTS
            none
         RESULTS
            proc-value
         NON-STANDARD CONTROL
            will generate an ABORT signal if no space available for
            new stack
         GLOBALS
            none
         %
      % Declarations %
         (stkptr);                                                8J1B1
         (constk);                                                8J1B2
         (temp);                                                  8J1B3
         (i);                                                     8J1B4
      % make sure a coroutine address was passed to us %
         IF NOT &acoroutine THEN
            ABORT( programbug, $"ALOSTK: no coroutine address
            passed" );
      % get a new stack of size nstksize + overhead words from
      specified zone %
         IF NOT &astk THEN
            IF NOT (&astk _ getblk( nstksize + stackdecsize, zone ))
            THEN
               ABORT( programbug, $"Insufficient room in zone for
               new stack");
      % initialize the stack descriptor for this new stack %
         &astk _ &astk + stackdecsize;
         A1 _ -nstksize; !HRL A1,A1; A1.RH _ &astk;
         stkptr _ A1;
         astk[stkmsave] _ astk[stkssave] _ astk[stksig] _
         astk[stkdalo] _ astk[stkmdalo] _ astk[stksdalo] _ 0;
         astk[stkzone] _ zone;
         astk[stksize] _ nstksize;
         IF ( constk _ astk[stklink] _ ([sysbtm.RH][stklink] :=
         stkptr) ) THEN
            BEGIN
            [constk.RH][stkblink] _ stkptr;
            END;
         astk[stkblink] _ sysbtm;
      % set linlink to point to first stack in chain %
         IF NOT linlink THEN
            BEGIN
            linlink _ $gstack;
            linlink.LH _ -gstksz;
            END;
```

```
% put first word on bottom of stack %
   astk _ $sysufl;
% save state of old stack (M, S, & signal status) %
   [sysbtm][stkssave] _ M - 2000002B;
   [sysbtm][stkmsave] _ [M];
   sigsave( [M-1] );
% indicate which stack we came from, and indicate where new
stack is %
   pcstack _ sysbtm := stkptr;
   sysse _ &astk + nstksize;
% set up signal status of new stack (no signals in progress) %
   sigrestore();
% move S to 1st entry of new stack %
   S _ stkptr;
% move arguments for coroutine to new stack in reverse order %
   WHILE (i _ 1) <= nargs DO
      BEGIN
      temp _ [$nargs - 1 - nargs + i];
      !PUSH S,temp;
      END;
% set CORETURN location in our caller ([[M]+1]) to be our
return location ([M-1]); in otherwords, make it look like our
caller did an OPENPORT to the new coroutine on the new stack .
 The coroutine location is one cell beyond the chain word of a
frame (pointed to by M for the current frame.)  It is NOT a
stack pointer, but rather has a right half which is a pointer
to code which presumably did an openport.  The word one cell
before the chain word is a return location if the frame is a
procedure.  If the frame is for a coroutine, the word is a
stack pointer tothe frame which openported it (if it has not
been stored away in a local).  This word is accessible by the
word PORT.%
   [[M.RH] + 1] _ [M.RH - 1];
   R6 %mreg% _ [M.RH];
% setup to OPENPORT coroutine %
   A1 _ &acoroutine;
% switch stacks; i.e., make new stack current %
   M _ stkptr;
% now OPENPORT new coroutine %
   !PUSHJ S,1(A1);
% we should never get here %
   ABORT( programbug, $"ALOSTK: System screwed up");
END.

%.Pes;%
```

```
(dalostk)   % deallocate stack %
PROCEDURE (astk);                                                    8J2
   % Procedure description
      FUNCTION
         A call on this procedure is logically equivalent to
         deallocates a stack; it propogates a note unwind through
         the entire stack, unchains it from the stack chain, then
         frees associated allocated storage.
      ARGUMENTS
         none
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         An ABORT signal is generated if an attempt is made to
         deallocate the default system stack, GSTACK;  also, if
         the storage allocator generates and error; also, if a
         stack tries to deallocate itself.
      GLOBALS
         none
      %
   % Declarations %
      (fptr);                                                        8J2B1
      (bptr);                                                        8J2B2
      (sigp) REF;                                                    8J2B3
      (endadr);                                                      8J2B4
      (catp) REF;                                                    8J2B5
   % Generate ABORT if this is GSTACK or if we stack is trying to
   deallocate itself or if stack is currently being dealocated by
   anyone else %
      IF [astk.RH] = $gstack THEN
         ABORT(programbug, $"Attempt to deallocate system
         stack.");
      IF [astk.RH] IN [sysbtm.RH, (sysbtm.RH-sysbtm.LH).RH] THEN
         ABORT(programbug, $"Attempt by stack to deallocate
         itself.");
      IF [astk.RH][stkdalo] THEN
         ABORT(programbug, $"Attempt to dealocate a dealocated
         stack.");
   % Propogate NOTE(unwind) down entire stack to permit cleanup %
      sigsave( M );
      R1.LH _ - [astk.RH][stksize];   R1.RH _ astk;
      astk _ R1; % to insure astk LH set %
      [R1.RH][stkdalo] _ sysbtm;
      [sysbtm.RH][stkssave] _ [R1.RH][stksdalo] _ S :=
      [R1.RH][stkssave];
      [sysbtm.RH][stkmsave] _ [R1.RH][stkmdalo] _ M :=
      [R1.RH][stkmsave];
      sysbtm _ R1;
      %sigrestore(); not actually done: we aren't interested in
      (other) nested signals on the stack being deallocated %
      NOTE(unwind);
      M _ [sysbtm.RH][stkmdalo];
      S _ [sysbtm.RH][stksdalo];
      sysbtm _ [sysbtm.RH][stkdalo];
      sigrestore();
   % Get rid of signals in progress associated with stack being
```

```
deallocated %
   FOR &sigp _ &sysgpt-sigsiz DOWN sigsiz UNTIL < $sysgstk DO
      BEGIN
      IF sigp[sgstk] = astk THEN % signal in stack being
      deallocated %
         BEGIN
         IF tsysgn THEN BUMP DOWN tsysgn;
         IF sysinfo >= &sigp THEN
            sysinfo _ sysinfo-sigsiz; % the global signal will
            move down the stack %
         IF &sigp = &sysgpt-sigsiz THEN
            &sysgpt _ &sysgpt -sigsiz
         ELSE
            BEGIN
            % check for underflow %
               IF (&sysgpt-sigsiz) < $sysgstk THEN
                  sysrcv(programbug,$"signal stack
                  underflow",0);
            &sysgpt _ &sysgpt - sigsiz;
            R4.LH _ &sigp + sigsiz;
            R4.RH _ &sigp;
            endadr _ &sysgpt;
            !BLT R4,@endadr;
            END;
         END;
      END;
% Drop catchphrases associated with stack being deallocated %
   FOR &catp _ $syscstk UP catsiz UNTIL >= &syscpe DO
      IF catp[catstk] = astk THEN
         drpcat _ catp[catenc] _ dropped;
   IF NOT tsysgn AND drpcat THEN sysdpall();
% Unchain stack %
   fptr _ [astk.RH][stklink];
   bptr _ [astk.RH][stkblink];
   IF bptr THEN [bptr.RH][stklink] _ fptr;
   IF fptr THEN [fptr.RH][stkblink] _ bptr;
   IF NOT gstack[stklink] THEN linlink _ 0;
% Deallocate %
   IF [astk.RH][stkzone] THEN
      BEGIN % stack was allocated %
      IF NOT freeblk([astk.RH]-stackdecsize,
      [astk.RH][stkzone]) THEN
         ABORT( programbug, $"Error on stack deallocation");
      END;
RETURN;
END.

%.Pes;%
```

```
% String Construction.Post=Off;%
(bsc) %Begin string construction.  Called with STID (or pointer to
string with stastr TRUE) of the destination.%
                                                                    9A
   PROCEDURE (dest);
   rplsid _ dest;
   %set up byte pointer and SAR for string construction%
   nsdbpt _ chbmty + $sar;
   sar.L _ empty;
   %reset stack for modified clist entries%
   clchng _ clmpty;
   RETURN END.

(esc) %End string construction.%
                                                                    9B
   PROCEDURE;
   IF rplsid.stastr THEN %A-string%
      BEGIN
      *[rplsid.stadr]* _ *sar*;
      RETURN;
      END;
   filesc();
   RETURN;
   END.

(kps) %argument points to string to be appended to the statement
being constructed%
                                                                    9C
   PROCEDURE (asfrom);
   REF asfrom;
   LOCAL bptr; %byte pointer for reading characters%
   IF (sar.L _ sar.L + asfrom.L) > sar.M THEN
      ABORT(saroverflow,$"string too long");
   bptr _ chbmty + &asfrom;
   a2 _ asfrom.L;
   a3 _ nsdbpt;
   a4 _ bptr;
   UNTIL (a2 _ a2-1) < 0 DO ^a3 _ a1 _ ^a4;
   nsdbpt _ a3;
   RETURN END.

(apachr) %passed character as argument.  appends to the statement
being constructed in sar.%
                                                                    9D
   PROCEDURE (ch);
   IF (sar.L _ sar.L+1) > sar.M THEN ABORT(saroverflow,$"string too
   long");
   ^nsdbpt _ ch;
   RETURN END.

(aptstr) %Append tstring.  Argument is pointer to two T-pointers
which delimit the substring to be appended to the statement
currently being constructed.%
                                                                    9E
   PROCEDURE (tp);
   LOCAL
```

```
    cl,      %location of record for a cl to be updated%
    end,     %location of end of cl table%
    cltab,   %location of cl table%
    flhd,    %file header loc%
    ch,      %character read from the tstring%
    bfrom,    %source byte pointer%
    lenstat,  %number of chars from string%
    lenblank; %number of chars from string%
REF cl;
IF [tp] # [tp+2] OR [tp+1] <= empty THEN
    BEGIN
    modeset _ 0; %reset global modeset flag %
    sysrcv(programbug,$"illegal text-pointer pair",sysclr());
    END;
IF [tp+1] >= [tp+3] THEN
    BEGIN
    modeset _ 0; %reset global modeset flag %
    RETURN;
    END;
INVOKE(catch);
stcwrk _ [tp];
stcwrl _ [tp+1];
techcl (forward, $stcwrk);
% stcwrk[2] now contains length + 1 of source string %
% calculate number of chars to take from source string %
CASE stcwrk[2] OF
    <= [tp+1] :
        BEGIN
        lenstat _ 0;
        lenblank _ [tp+3] - [tp+1];
        END;
    < [tp+3] :
        BEGIN
        lenstat _ stcwrk[2] - [tp+1];
        lenblank _ [tp+3] - stcwrk[2];
        END;
    ENDCASE
        BEGIN
        lenstat _ [tp+3] - [tp+1];
        lenblank _ 0;
        END;
&cl _ cltab _ [clstad].clbuff;
    %address of start of clist%
end _ &cl + [clstad].clcnt * cll;
    %end of clist%
DROP(catch);
IF modeset THEN
    BEGIN % mode set now in operation %
    modeset _ 0; %reset the global flag%
    UNTIL (lenstat _ lenstat-1) < 0 DO
        BEGIN
        ch _ READC($stcwrk);
        %now append the character%
        CASE modeshift OF %check for forced case%
            =0: apachr(IF ch IN ['a,'z] THEN ch-40B ELSE ch);
                %forced upper case%
```

```
            =1: apachr(IF ch IN ["A,"Z] THEN ch+40B ELSE ch);
               %forced lower case%
            ENDCASE apachr(ch); %no case change%
         END;
      END
   ELSE %no mode change. just copy across%
      BEGIN
      IF NOT mkrstay THEN
         IF NOT rplsid.stastr THEN
            BEGIN
            IF rplsid.stfile = [tp].stfile THEN
               UNTIL &cl = end DO
                  BEGIN
                  IF cl.clst1 = [tp] AND
                     cl.clcc1 IN [[tp+1],[tp+3]) AND
                     NOT cl.clfixed THEN
                        BEGIN
                        cl.clfixed _ TRUE;
                        cl.clcc1 _
                           sar.L + 1 + cl.clcc1 - [tp+1];
                        PUSH &cl ON clchng;
                        END;
                  &cl _ &cl + cll;
                  END
            ELSE cldsr(tp);
            END;
      IF (sar.L _ sar.L + lenstat) > sar.M THEN
         ABORT(saroverflow,$"string too long");
      a4 _ stcwrk[4];  %byte pointer made by fechc1%
      a2 _ lenstat;
      a3 _ nsdbpt;
      UNTIL (a2 _ a2-1) < 0 DO ^a3 _ a1 _ ^a4;
      nsdbpt _ a3;
      END;
   UNTIL (lenblank _ lenblank-1) < 0 DO apachr(SP);
   mkrstay _ FALSE;
   RETURN;
   (catch) CATCHPHRASE;                                          9E21
      % make sure modeset is zero if the world gets away from us %
      BEGIN
      IF SIGNAL=unwind  THEN
         modeset _ 0;
      CONTINUE;
      END;
   END.

(cldsr) %Called with address of two tpointers.  For all clist
entries, if between these two pointers, then set to "deleted".%
                                                                 9F
   PROCEDURE (tp);
   LOCAL end, cl;
   REF cl;
   &cl _ [clstad].clbuff;
      %address of start of clist%
   end _ &cl + [clstad].clcnt * cll;
      %end of clist%
```

```
      UNTIL &cl >= end DO
         BEGIN
         IF cl.clst1 = [tp] AND
            cl.clcc1 IN [[tp+1],[tp+3]) AND
            NOT cl.clfixed THEN
            BEGIN
            cl.clst2 _ cl.clst1 := endfil;
            cl.clcc2 _ [tp+1];
            END;
         &cl _ &cl + cll;
         END;
      RETURN END.

% Reading characters from SDB%
   (fechc1)                                                    10A
      %Documentation%
         %This routine is called to initialize a work area for reading
         characters from a statement.  Arguments are: direction of
         reading characters (=0 then backwards) and the address of the
         7 word work area (of which the last 2 words are no longer
         used).
         If characters are to be read from a statement then
            when calling FECHC1, the first two cells of the work area
            must contain a Tpointer.  A character count of one
            indicates the first character of the statement. FECHC1 will
            initialize the rest of the work area. The first word of the
            word area always has the PSID of the statement.  The second
            word has the character count.  The third word contains a
            bound on characters to be read.  ENDCHR's are returned
            after reach this bound.  The fourth word has the direction
            of readout for use by readc.  A READC(x) actually results
            in the value of x being loaded into register wa followed by
            a JSP a4,readc.
            The fifth word of the work area contains a byte pointer to
            the character last read from the statement. Thus an ILDB
            instruction may be used to get the next character if the
            direction is forward.  If the direction of reading is
            backward. then the byte pointer is decremented by in-line
            code.
            To read characters from the statement execute a READC(x)
            where the value of x is the address of the work area to be
            used.  The character is returned as the value of the READC.
             Subsequent READC's will return the following characters.
            To change position or direction within the statement the
            work area must be reinitialized by calling FECHC1 again, as
            described above.  There may however be more than one work
            area currently in use, and these may be changed
            independently.
         If characters are to be read from an A-string then
            the first word of the work area contains the address of the
            A-string instead of a PSID.  The second word is 1 if the
            first character of the string is to be read next, two if
            the second, etc.  Characters may be read out of an A-string
            in either direction, just like a statment.  Endcharacters
            are returned when the string is exhausted.%
      PROCEDURE (dir, worka);
```

```
   LOCAL
      wp,   %word postion of the starting character%
      cp,   %character postion of the starting character%
      stdb,  %stdb for statement%
      rng,  %location of ring element%
      addr; %address of the word containing starting character%
   REF rng;
   %set work+2 to bound%
   IF [worka] = endfil THEN %set to null string%
      BEGIN
      [worka+3] _ 2; %readc will return ENDCHR%
      RETURN;
      END
   ELSE IF [worka].stastr THEN %A-string%
      BEGIN
      addr _ [worka].stadr;
      [worka+2] _ IF pe THEN pe ELSE [addr].L + 1;
      addr _ addr + 1;
      END
   ELSE %SDB%
      addr _ flfechc1( dir, worka : stdb );
   %find word and character position%
   IF dir THEN %scan forward%
      BEGIN
      [worka+3] _ 1; %direction%
      END
   ELSE %scan backwards%
      BEGIN
      [worka+2] _ IF ps THEN ps ELSE 1; %change bound%
      [worka+3] _ 0; %direction%
      END;
   %make byte pointer to the previous (forward) or current
   (backward) character%
   DIV ([worka+1]-[worka+3])/5, wp, cp;
   [worka+4] _ 44070086 - cp * 7000086 + addr + wp;
   RETURN END.

(openswork) PROCEDURE;                                          10B
   fechc1(1, $swork);
   RETURN END.

(savpos) PROCEDURE;                                             10C
   PUSH ps ON btwstk;
   PUSH pe ON btwstk;
   RETURN END.

(respos) PROCEDURE;                                             10D
   POP btwstk TO pe;
   POP btwstk TO ps;
   RETURN END.

(xxreadc) PROCEDURE; %This is the routine that is called to read a
character by the READC construct.  Code to call is JSP a4,readc so
return loc is in A4.%                                           10E
   (readc):                                                     10E1
   %cant have locals since is not called by usual procedure linkage.
```

```
     wa is a register containing the address of the work area.%
    !SKIPG a1,3(wa);
       !JRST readc1;
    !SOJN a1,readend;
    %forward scan%
       !AOS a1,1(wa);
       !CAMLE a1,2(wa);
          !JRST rdend1;
       !ILDB a1,4(wa); %get the character%
       !JRST 0(a4);
    (readc1):                                                    10E6
    !JUMPN a1,readend;
    %backward scan%
       !SOS a1,1(wa);
       !CAMGE a1,2(wa);
          !JRST rdend2;
       %back up byte pointer%
       !MOVE a1,4(wa);
       !ADD a1,=7B10;
       !CAIG a1,0;
       !SUB a1,=4300000000018;
       !MOVEM a1,4(wa);
       !LDB a1,a1; %get the character%
       !JRST 0(a4);
    (rdend2): !AOSA 1(wa); %adjust for backward overrun, NOTE SKIP%
                                                                 10E9
    (rdend1): !SOS 1(wa); %adjust for forward overrun, SKIPPED OVER%
                                                                10E10
    (readend): %out of bounds%                                  10E11
       a1 _ ENDCHR;
       !JRST 0(a4);
    END.

% A-string routines%
   (asrref) %create a pointer to astring with stastr field set.%
                                                                 11A
       PROCEDURE (ast);
       ast.stastr _ TRUE;
       RETURN (ast) END.

   DECLARE % byte pointers for chbptr %
       chparray=(
          350700000001B,
          260700000001B,
          170700000001B,
          100700000001B,
          107000000018);

   DECLARE cshift=(7,14,777761B %-15%, 777770B %-8%, 777777B %-1%);
   DECLARE ascomm=(0,774B9,77776B7,7777777b5,-4008);
   EXTERNAL cmpstr, comstr, apstore; % inter-procedure refs %
   (idchr) % load character from a-string %                      11F
       PROCEDURE
          (astr, % a-string address %
          charno); % index, 1=first char %
       % return a specified character from an astring %
```

```
      !MOVE A1,charno; % get index %
      !MOVEI A1,-1(A1);
      !IDIVI A1,5;  % char-1/5 %
      !ADD A1,astr; % add address of string %
      !MOVE A1,1(A1);  % get word from string %
      !ROT A1,@cshift(A2);  % rotate and mask %
      !ANDI A1,177B;  % faster than LDB %
      RETURN;
   END.
(chbptr) % return a-string char byte-pointer given index %          11G
   PROCEDURE
      (charno); % character index 1=first char  if LDB %            11G1A
         % NOTE: 0=first char if do ILDB on resulting pointer %
   % return a partial byte pointer given a character number.  The
   resulting byte pointer needs to have the address of the astring
   added to it %
      !SKIPG A1,charno;  %check for zero char count %
      !JRST chbpt1;
      !MOVEI A1,-1(A1);  %subtract one %
      !IDIVI A1,5;
      !ADD A1,chparray(A2);  %get partial pointer %
      RETURN;
      (chbpt1):                                                     11G2G
         RETURN(chbmty);
   END.
(apchr) % append a character to a-string %                          11H
   PROCEDURE
      (char, % the character to append %
      astr); % the a-string address %
   % Append the given character to the specified astring.  If the
   string is at it's max length a HELP(stringoverflow,astr) results
   %
      LOOP
         BEGIN
         A2 _ [astr];  % get max,,len from string %
         !MOVSI A1,1(A2); % get len+1 %
         !CAMG A1,A2;  % over max ? %
         EXIT LOOP;
         astr _ syssov(astr, astr.L+1); % get new string %
         END;
      !HLRM A1,@-3(M);  % store len+1 %
      !MOVEI A2,0(A2);  % isolate new length-1 = old len %
      (apstore):                                                    11H2D
         !IDIVI A2,5;
         !ADD A2,chparray(A3);  % get partial pointer %
         !ADD A2,astr;  % plus string address %
         !MOVE A1,char;  % get character %
         !DPB A1,A2;  % and store it %
         RETURN;
   END.
(stbptr) % non-astring character byte pointer%                      11I
   PROCEDURE (charno);
   %creates a partial byte pointer to the character determined by
   the character number passed as argument.  The resulting byte
   pointer needs to have the address of the string (NOT ASTRING)
   added to it.%
```

```
    LOCAL wp, cp;
    IF charno = empty THEN RETURN (440700000000B);
    DIV (charno-1) / 5, wp, cp;
    RETURN (chparray[cp] + wp - 1) END.

(ascom) PROCEDURE(astr1, astr2, relation);                          11J
    % accepts addresses of 2 a-strings.  returns logical value of
    astr1 <relation> astr2 %
    LOCAL reltabi;
    r1 _ -1;  % set flag in r1 - means must do all of compare %
    r2 _ relation;
    (comstr):  % called by ascom and compas %                       11J5
        % r1 is a flag: =0 means return false if a mismatch (compas
        call). <0 means must compare entire string and return reltab
        (ascom call). >0 means return the sign of astr1 = astr2
        (cmpstr call) %
        IF [astr1].L # [astr2].L THEN BEGIN
            !JUMPE r1,ascom6;  % RETURN(FALSE) if called from compas %
            !SKIPE lngflg;  % cmpstr: IF lngflg AND lengths not equal %
            !JUMPG r1,ascom5;  % cmpstr call - compare lengths %
            !CAIN r2,2;  % is relation = ? %
            !JRST ascom6;  % yes.  RETURN(FALSE) %
            !CAIN r2,6;  % if relation # ? %
            !JRST ascom7  % yes.  RETURN(TRUE) %
            END;
        !IMULI r2,3;  % r2 _ r2*3+1 %
        !MOVEI r2,1(r2);  % plus one %
        a3 _ [m-2];  % $astr1 %
        a4 _ [m-3];  % $astr2 %
        !HRRZ a1,0(a3);  % len of astr1 %
        !HRRZ a2,0(a4);  % len of astr2 %
        !CAIG a2,0(a1);  % find min len %
        !MOVEI a1,0(a2);  % now a1 has the min %
        !IDIVI a1,5;  % length/5: a1=number of words, a2=mask index %
        (ascom2):                                                   11J5L
            !MOVEI a3,1(a3);  % add one to pointer %
            !MOVEI a4,1(a4);  % ditto %
            !MOVE r3,0(a3);  % get a word of string %
            !TRZ r3,1;  % turn off low order bit %
            !SOJL a1,ascom1;  % last word of string ? %
            !MOVE r4,0(a4);  % get other word %
            !TRZ r4,1;  % and turn of low order bit %
            !SUB r3,r4;  % now, take difference %
            !JUMPE r3,ascom2;  % if equal, keep going %
            !JUMPE r1,ascom6;  % quit here if we can (i.e. from compas)
            %
        (ascom3):                                                   11J5M
            !TLNE r3,400005;  % compute sign of result:  sign bit on? %
            !SKIPA r3,=-1;  % yes, get -1 %
            !MOVEI r3,1;  % no, get one %
        (ascom4):                                                   11J5N
            !JUMPG r1,ascom8;  % cmpstr call - return sign, not reltab
            %
            !ADDI r3,0(r2);  % add in reltab index %
            RETURN(reltab[r3]);  % return result from table %
        (ascom1):                                                   11J5O
```

```
        !JUMPE a2,ascom5;  % last word of str.  any chars in it? %
        !AND r3,ascomm(a2);  % and off extraneous chars %
        !MOVE r4,0(a4);  % get word from other string %
        !AND r4,ascomm(a2);  % and it also %
        !SUB r3,r4;  % take difference %
        !JUMPN r3,ascom3;  % check sign if not equal %
      (ascom5):                                                    11J5P
        !HRRZ r3,@-2(m);  %last words equal.  compare lengths %
        !HRRZ r4,@-3(m);  % get other length %
        !SUBI r3,0(r4);  % take difference %
        !JUMPE r3,ascom4;  % equal, return result %
        !JRST ascom3;  % not equal, compute sign %
      (ascom6):                                                    11J5Q
        RETURN(FALSE);
      (ascom7):                                                    11J5R
        RETURN(TRUE);
      (ascom8):                                                    11J5S
        RETURN(r3); % return sign for cmpstr call %
    NULL END.
(compas) PROCEDURE(astr1, astr2);                                  11K
    % compares the contents of two a-strings.  Returns true if the
    contents match, false otherwise %
    r1 _ 0; r2 _ 2; % relation = true %
    GOTO comstr END.
(cmpstr) PROCEDURE(astr1, astr2);                                  11L
    % return -1 if astr1 < astr2, 0 if equal, 1 if astr1 > astr2 %
    r1 _ 1; % set flag >0 means return sign value %
    R2 _ 0; % condition cannot be 2 or 6 %
    GOTO comstr END.
(repchr) %replace a character in an astring%
                                                                   11M
    PROCEDURE (char, ast, chrno);
    %------------------%
    LOCAL cnt;
    REF ast;
    !MOVE a2,chrno; % the char number %
    !MOVEI a2,-1(a2);  % minus one %
    !JRST apstore
    END.

(apblnk) PROCEDURE(ast, cnt);                                      11N
    % Append cnt blanks to the designated astring. %
    REF ast;
    FOR cnt DOWN UNTIL < 1 DO *ast* _ *ast*, SP;
    RETURN END.

(apsr) %The purpose of this routine is to append one A-string onto
another.  Arguments are two A-string addresses.  The first string is
appended to the second.  A HELP signal (stringoverflow) is generated
if the maximum length of the latter string is too short to contain
the result.%
                                                                   11O
    PROCEDURE (asfrom, asto);
    LOCAL
        nlen, % new string length %
        wp,    % word position %
```

```
      cp,    % character position %
      bfrom, % byte pointer in source string %
      bto;   % byte pointer in destination string %
   REF asfrom, asto;
   IF (nlen _ asto.L + asfrom.L) > asto.M THEN
      &asto _ syssov(&asto, nlen); % get help %
   bfrom _ chbmty + &asfrom;
   bto _ chbptr(asto.L) + &asto;
   asto.L _ nlen;
   a2 _ asfrom.L;
   UNTIL (a2 _ a2-1) < 0 DO ^bto _ a1 %use a1% _ ^bfrom;
   RETURN END.

(cpysr) %To copy one A-string into another, call this routine with
the address of the source string and the address of the destination
string (in that order).  An ERROR is generated if the maximum length
of the destination string is shorter than the source string.%
                                                                  11P
   PROCEDURE (asfrom, asto);
   REF asfrom, asto;
   CASE asfrom.L OF
      > asto.M :
         BEGIN
         &asto _ syssov(&asto, asfrom.L); % get help %
         REPEAT CASE;
         END;
      <= empty : asto.L _ empty;
      ENDCASE
         BEGIN
         asto.L _ asfrom.L;
         mvbfbf(&asfrom+1, &asto+1, (asfrom.L+4)/5);
         END;
   RETURN END.

(chpair) %extract substring and append to another string%          11Q
   PROCEDURE(ast1, lower, upper, ast2);
   LOCAL nlen, bto, bfrom, lenstr, lenblank;
   REF ast1, ast2;
   IF upper < lower THEN RETURN;
   lower _ MAX(1, lower);
   CASE ast1.L OF
      < lower :
         BEGIN
         lenstr _ 0;
         lenblank _ upper-lower+1;
         END;
      < upper :
         BEGIN
         lenstr _ ast1.L-lower+1;
         lenblank _ upper-ast1.L;
         END;
      ENDCASE
         BEGIN
         lenstr _ upper-lower+1;
         lenblank _ 0;
         END;
```

```
   LOOP
      IF (nlen _ ast2.L + lenstr + lenblank ) <= ast2.M THEN EXIT
      ELSE &ast2 _ syssov(&ast2, nlen); % get help %
   bto _ chbptr(ast2.L) + &ast2;
   bfrom _ chbptr(lower-1) + &ast1;
   ast2.L _ nlen;
   a2 _ lenstr;
   a3 _ bto;
   a4 _ bfrom;
   UNTIL (a2 _ a2 - 1) < 0 DO ^a3 _ a1 _ ^a4;
   a2 _ lenblank;
   a1 _ SP;
   UNTIL (a2 _ a2 - 1) < 0 DO ^a3 _ a1;
   RETURN END.

(syssov) % string overflow - get HELP routine %                    11R
   % attempt to get new string address from routine up in thread of
   control.  If get it, broadcast new address via NOTE and return
   it.  Otherwise ABORT. %
   PROCEDURE
      (astr, % A-string address of too-short string %
      nlen); % required string length (total length) %
   LOCAL nastr; % new astring address %
   IF HELP(stringoverflow,astr, nlen: nastr) = gothelp THEN
      NOTE(changestring,astr,nastr)
   ELSE ABORT(stringoverflow,$"string too long");
   RETURN(nastr);
   END.
(mvbfbf) %This routine moves a buffer of words.  Arguments are
address of source, address of destination, and number of words to
transfer.  It returns the address of the last word into which data
was moved.%
                                                                   11S
   PROCEDURE (bfrom, bto, nw);
   LOCAL lw;
   IF nw = 0 THEN RETURN;
   lw _ nw + bto - 1; %last word to be transferred to%
   IF bto IN (bfrom, bfrom+nw) THEN
      BEGIN
      a1 _ bfrom;
      a2 _ nw; a3 _ a2;
      a2 _ a2 + a1; %pointer into source%
      a3 _ a3 + bto; %pointer into destination%
      UNTIL (a2 _ a2-1) < a1 DO
         BEGIN
         a3 _ a3 - 1;
         [a3] _ [a2];
         END;
      END
   ELSE %can use block transfer instruction%
      BEGIN
      !HRL a1,bfrom; !HRR a1,bto; !BLT a1,@lw
      END;
   RETURN (lw) END.

DECLARE hshmsk=(
```

```
    774B9,
    77776B7,
    777777B5,
    7777777774B2,
    77777777776B);
```

(hash) %An a-string address as argument. The hash code for that string is returned.%

11U

```
    PROCEDURE (ast);
    LOCAL nw, i, gen1, gen2, oldl, cp;
    REF ast;
    IF (oldl _ ast.L) <= empty THEN RETURN(0);
    DIV (oldl + 4)/5, nw, cp;
    FOR i _ 1 UP UNTIL = nw DO ast[i] _ ast[i] .A hshmsk[4];
    ast[nw] _ ast[nw] .A hshmsk[cp];
    gen1 _ 0;
    gen2 _ 1;
    UNTIL nw = 0 DO
        BEGIN
        gen1 _ ast[nw] * gen2 + gen1 / 17;
        gen2 _ gen2 * 43;
        nw _ nw - 1;
        END;
    a1 _ gen1;
    !LSH a1,-6;
    RETURN(a1) END.
```

(astruc) PROCEDURE(astrng); %a-string to upper case% 11V

```
    % convert a-string (in astrng) to upper case %
    %----------------%
    LOCAL length, bytptr, char;
    REF astrng;
    IF (length _ astrng.L) = empty THEN RETURN(&astrng);
    bytptr _ chbptr(empty) + &astrng;
    DO IF (char _ ^bytptr) IN ["a,"z] THEN
        .bytptr _ char - 40B
    UNTIL (length _ length - 1) = empty;
    RETURN(&astrng);
    END.
```

(singth) PROCEDURE (bp1, bp2); %string length% 11W

```
    %returns the lenght of the string  represented by the passed byte
    pointers, the first of which is assumed to be set so that an
    increment and load byte will get first char of the string and the
    second pointing to the last char in the string.  NOTE: the index
    field is ignored and the only the size field of bp1 is used!%
    %bpadr, bpsize, bpbitpos%
    %----------------%
    LOCAL length;
    length _ (bp2.bpadr - bp1.bpadr)*(36 / bp1.bpsize) +
        (bp1.bpbitpos - bp2.bpbitpos) / bp1.bpsize;          11W5A
    RETURN(MAX(length, 0));
    END.
```

(srmake) %make string from value% 11X

```
    PROCEDURE(value,astrng,base,format);
    LOCAL char, bpntr, b2pntr, b3pntr, mag, psign, ovrflw, cp, npad,
    nchars, i, sign, ncols, rcols, fill, decml;
    LOCAL STRING locstr[40];
    REF astrng;
    % make sure no floating point to confuse things %
       cp _ ovrflw _ npad _ 0;
    % initialize byte pointers %
       bpntr _ b2pntr _ $locstr + 1 .V 440700B6;
    % extend sign of RH if requested %
       IF (format.fmsgne := FALSE) THEN !HRRES value;
    % setup for nout %
       mag _ IF (format.fmlgcl := FALSE) THEN 4B11 ELSE 0;
       psign _ IF (format.fmpsgn := FALSE) THEN 2B11 ELSE 0;
    % get number string (& do nothing if bad radix passed) %
       IF NOT SKIP !nout( bpntr, value, base .V mag .V psign) THEN
       RETURN
       ELSE nchars _ locstr.L _ slngth( bpntr, R1 );
    % pick up sign char (if any) and adjust pointer %
       CASE sign _ ^b2pntr OF
          = "+, = "-:
             BEGIN
             BUMP DOWN nchars;
             bpntr _ b2pntr;
             END;
          ENDCASE sign _ FALSE;
    (srovr):                                                    11X11
    % pick up other parameters & compute # needed columns %
       ncols _ locstr.L + (decml _ format.fmdecm);
       IF NOT (rcols _ format.fmncols) THEN rcols _ ncols;
       fill _ format.fmjstfy;
    % compute number of padding characters required %
       npad _ rcols - ncols;
    % check for overflow %
       IF npad < 0 THEN
          CASE format.fmovrf OF
             = 0:  RETURN;     % do notihing %
             = 1, = 5:         % MSDs only (& optionally a *) %
                BEGIN
                nchars _ nchars + npad;
                IF nchars <= 0 THEN REPEAT CASE( 4 );
                locstr.L _ locstr.L + (npad := 0);
                *locstr*[locstr.L + 1] _ 0;
                IF format.fmovrf = 5 THEN *locstr*[locstr.L] _ "*;
                END;
             = 2, = 6:         % (optionally a * &) LSDs only %
                BEGIN
                nchars _ nchars + npad;
                IF nchars <= 0 THEN REPEAT CASE( 4 );
                b3pntr _ b2pntr _ bpntr;
                FOR i _ -npad DOWN UNTIL = 0 DO char _ ^b3pntr;
                FOR i _ (nchars + 1) DOWN UNTIL = 0 DO
                   ^b2pntr _ ^b3pntr;
                IF format.fmovrf = 6 THEN ^bpntr _ "*;
                locstr.L _ locstr.L + (npad := 0);
                END;
```

```
            = 3:     % spaces in entire field %
               BEGIN
               WHILE (rcols := rcols - 1) DO
                  IF &astrng THEN *astrng* _ *astrng*, SP
                  ELSE apachr( SP );
               RETURN;
               END;
            = 4:     % *s in entire field %
               BEGIN
               WHILE (rcols := rcols - 1) DO
                  IF &astrng THEN *astrng* _ *astrng*, "*
                  ELSE apachr( "* );
               RETURN;
               END;
            ENDCASE;
   % leading spaces %
      IF fill AND (NOT format.fmfill) THEN srm3( &astrng, npad := 0,
      SP);
   % sign if required %
      IF sign THEN
         BEGIN
         IF &astrng THEN *astrng* _ *astrng*, sign
         ELSE apachr( sign );
         cp _ 1;
         END;
   % leading zeros %
      IF fill AND format.fmfill THEN srm3( &astrng, npad := 0, "0);
   % significant digits %
      srm1( &astrng, $locstr, cp );
   % trailing zeros %
      IF format.fmfill THEN srm3( &astrng, npad := 0, "0);
   % terminating decimal %
      IF decml THEN
         IF &astrng THEN *astrng* _ *astrng*, "."
         ELSE apachr(".");
   % final spaces %
      srm3( &astrng, npad, SP);
   RETURN END.

(srm3) PROCEDURE( astrng REF, npad, pchar);                          11V
   WHILE (npad := npad-1) > 0 DO
      IF &astrng THEN *astrng* _ *astrng*, pchar
      ELSE apachr( pchar );
   RETURN END.

(srm1) PROCEDURE( astrng REF, str2 REF, cp);                         11Z
   LOCAL TEXT POINTER tp1, tp2;
   IF &astrng THEN *astrng* _ *astrng*, *str2*[cp+1 TO str2.L]
   ELSE
      BEGIN
      tp1 _ tp2 _ &str2;
      tp1.stastr _ tp2.stastr _ TRUE;
      tp1[1] _ cp+1;
      tp2[1] _ str2.L + 1;
      aptstr( $tp1 );
      END;
```

```
   RETURN END.

(srmk) PROCEDURE(value,base, format);                                  11A@
   %In string construction use apachr to append string to sar.%
   srmake(value, 0, base, format);
   RETURN END.

(srval) PROCEDURE(astrng, base);                                       11AA
   %convert string to value%
   LOCAL value, cnt, char;
   REF astrng;
   value _ 0;
   cnt _ 1;
   UNTIL cnt > astrng.L DO
      BEGIN
      char _ *astrng*[cnt];
      char _ (CASE char OF
         IN [ "0, "9]: char - "0;
         IN ["A, "Z]: char - "A + 10;
         IN ["a, "z]: char - "a + 10;
         ENDCASE char - "0);
      value _ value*base + char;
      BUMP cnt;
      END;
   RETURN (value) END.

DECLARE reltab = (0,0,0, %no 0 relational%
   TRUE, FALSE, FALSE, %1: <%
   FALSE, TRUE, FALSE, %2: =%
   TRUE, TRUE, FALSE, %3: <=%
   0, 0, 0, %no 4 relational%
   FALSE, TRUE, TRUE, %5: >=%
   TRUE, FALSE, TRUE, %6: #%
   FALSE, FALSE, TRUE); %7: >%

(repstr) %The routine replaces the characters in one a-string,
OLDAST, with those in another, NEWAST, beginning at CHARNO. If the
new a-string is empty, the routine returns FALSE; otherwise it
returns TRUE.%
                                                                       11AC
   PROCEDURE (newast, oldast, charno);
   %------------------%
   LOCAL astrl, repcnt;
   REF newast, oldast;
   IF (astrl _ newast.L) = empty THEN RETURN (FALSE);
   IF charno + newast.L > oldast.L THEN
      BEGIN
      IF charno + newast.L > oldast.M THEN
         RETURN (FALSE);
      oldast.L _ charno + newast.L;
      END;
   repcnt _ empty + 1;
   UNTIL repcnt > astrl DO
      BEGIN
      *oldast*[charno] _ *newast*[repcnt];
      BUMP charno, repcnt;
```

```
        END;
    RETURN (TRUE)
    END.

(srlset)  %To set an A-string to a single character, call this
procedure with the character and the address of the string.%
                                                                11AD
    PROCEDURE (char, ast);
    [ast].L _ empty;
    apchr (char, ast);
    RETURN END.

(mkbptr) %Make Byte Pointer with the specified position, size and
address.  The address is 23 bits wide to allow indexing and
indirection.%
                                                                11AE
    PROCEDURE (pos, size, addr);
    a1 _ addr; !LSHC a1,-24;
    a1 _ size; !LSHC a1,-6;
    a1 _ pos;  !LSHC a1,-6;
    RETURN (a2) END.

% Statement scanning & cntent-analysis support%
(pdc) PROCEDURE (pdcnum, pntloc);                               12A
    %pointer decrement. arguments are number of postions to
    move and address of pointer to be decremented.  %

    LOCAL
        end;    %count at end of statement%
    IF scndir = forward THEN
        [pntloc+1] _ MAX([pntloc+1]-pdcnum, 1)
    ELSE
        BEGIN
        cpfse([pntloc]);
        end _ a2;
        [pntloc+1] _ MIN([pntloc+1]+pdcnum, end);
        END;
    RETURN;
    END.

(dptr) PROCEDURE(pntloc);                                       12B
    pdc(1, pntloc);
    RETURN END.

(tstsr) PROCEDURE (ast);                                        12C
    %Test string.  Compare the T-string specified by SWORK with the
    string whose address is passed as arg.  Fetches characters from
    string by calling READC.  On a successful match does RETURN TRUE
    and SWORK is left pointing to the character after the pattern.
    If the match fails, FALSE return.%

    LOCAL
        cnt,    %counter for characters in test string%
        char,   %character from source string%
        tsp;    %pointer into test string%
    REF ast;
```

```
   cnt _ ast.L;
   tsp _ chbmty + &ast;
   UNTIL (cnt _ cnt-1) < 0 DO
      IF (char _ READC) = ENDCHR OR ^tsp # char THEN RETURN [FALSE];
   RETURN [TRUE] END.

(tstf) PROCEDURE (ast);                                          12D
   %Like tst except looks for any occurence of the test string in
   the remaining portion of the T-string.  This is done by first
   scanning thru looking for the first character of the string, then
   if find it the rest of the string is?compared.  If get a mismatch
   SWORK is reset to character?after the start of the current
   partial match and the process is repeated.  Failure occurs only
   when the T-string is exhausted.  Returns with RETURN TRUE on
   success, FALSE return on failure.%

   LOCAL
      cnt,     %counter for characters in test string%
      cpos,    %value of swork1 where got match%
      nchar,   %number of characters in test string%
      char,    %character from source string%
      char1,   %first character of test string%
      tsp,     %pointer into the test string%
      tsp1;    %pointer to first character of test string%
   REF ast;
   nchar _ ast.L;
   IF nchar < 1 THEN RETURN [TRUE];
   tsp1 _ chbmty + &ast;
   char1 _ ^tsp1; %first character of string%
   LOOP %scan until get match for first character in string%
      CASE READC OF
         = ENDCHR : EXIT; %have reached the end%
         = char1 : %have a match on the first character%
            BEGIN
            cpos _ swork1;
            cnt _ 1;
            tsp _ tsp1;
            LOOP
               BEGIN
               IF (cnt _ cnt+1) > nchar THEN RETURN [TRUE];
               IF (char _ READC) = ENDCHR THEN EXIT 2;
               IF ^tsp # char THEN EXIT;
               END;
            %failure -- back to scanning for first character%
            swork1 _ cpos;
            fechc1(scndir, $swork);
            END;
         ENDCASE;
   RETURN [FALSE];
   END.

(bfs) %branch false and scan %
                                                                 12E
   % Resets the character position, then reads another character
   from the statement.  If this is not an ENDCHR the character
   number is put on the stack and control is transfered to the
```

```
   location specified in the address of the pop.%

      % The above is such an ancient statement, it has been left as
      is. (The PDP-10 has UUO's but the SDS 940 had POPs -- WHP got
      carried away). Now, if the char is not an ENDCHR, the number
      is put on the stack and the procedure does a failure return.
      The compiler puts out a JUMPE R6, to the address in question
      (the beginning of the floating scan). %
   PROCEDURE;
   swork _ [p-1];
   swork1 _ [p];
   techc1(scndir, $swork);
   IF READC # ENDCHR THEN
      BEGIN
      [p] _ swork1;
      RETURN[FALSE];
      END;
   p _ p - 2000002B;
   RETURN END.

(fxswork) PROCEDURE;                                                  12F
   techc1(scndir, $swork);
   RETURN END.

(srsup) PROCEDURE; %dummy procedure for string support code%          12G
   %these are all called by JSP a4,x and return by JRST (a4)%
   (fcp):                                                             12G2
      swork _ a1;
      swork1 _ a2;
      !JRST (a4);
   (pcp):                                                             12G3
      !PUSH p,swork;
      !PUSH p,swork1;
      !JRST (a4);
   (sptr):                                                            12G4
      !MOVE  a2, swork;
      !MOVEM a2,0(a1);
      !MOVE  a2, swork1;
      !MOVEM a2,1(a1);
      !JRST (a4);
   (begarb):                                                          12G5
      !PUSH p,swork;
      !PUSH p,swork1;
      !HLRZ a2,a1; %lower bound%
      !PUSH p,a2;
      !HRRZ a2,a1; %upper bound%
      !PUSH p,a2;
      a2 _ 0; %count%
      !PUSH p,a2;
      !JRST (a4);
   (fxsp1): %leave a1 and a2 unchanged%                               12G6
      sptr1 _ a1;
      a3 _ 1;
      sptr1[a3] _ a2;
      !JRST (a4);
   (fxsp2):                                                           12G7
```

```
      sptr2 _ a1;
      a3 _ 1;
      sptr2[a3] _ a2;
      !JRST (a4);
   (ipr):                                                    12G8
      a1 _ [a2];
      a2 _ [a2+1];
      !JRST (a4);
   END.

EXTERNAL fcp, pcp, sptr, begarb, fxsp1, fxsp2, lpr;

(incarb) PROCEDURE;                                          12I
   %update position on stack where last succeeded%
   [p-4] _ swork;
   [p-3] _ swork1;
   %increment count and compare to upper bound%
   RETURN(([p] _ [p]+1) < [p-1]) END.

(endarb) PROCEDURE;                                          12J
   p _ p - 3000003B;
   scp();
   p _ p - 2000002B;
   RETURN([p+5] IN [[p+3],[p+4]]) END.

(scp) PROCEDURE;                                             12K
   swork _ [p-1];
   swork1 _ [p];
   fechc1(scndir, $swork);
   RETURN END.

(scnlft) PROCEDURE;                                          12L
   fechc1(scndir _ 0, $swork);
   RETURN END.

(scnrht) PROCEDURE;                                          12M
   fechc1(scndir _ 1, $swork);
   RETURN END.

(chrct) PROCEDURE (char, chrcls); %Character class test%
                                                             12N
   CASE chrcls OF
      =1: %CH%
         IF char IN [0,177B] THEN GOTO cctyes;
      =4: %LD%
         IF char IN ["A,"Z] OR
            char IN ["a,"z] OR
            char IN ["0,"9] THEN GOTO cctyes;
      =11: %NLD%
         IF char # ENDCHR AND
            char NOT IN ["A,"Z] AND
            char NOT IN ["a,"z] AND
            char NOT IN ["0,"9] THEN GOTO cctyes;
      =7: %L%
         IF char IN ["a,"z] OR
            char IN ["A,"Z] THEN GOTO cctyes;
```

```
      =8: %D%
         IF char IN ['0,'9] THEN GOTO cctyes;
      =9: %PT%
         IF char IN [41B,174B] THEN GOTO cctyes;
      =10: %NP%
         IF char # ENDCHR AND
            char NOT IN [41B,174B] THEN GOTO cctyes;
      =5: %UL%
         IF char IN ['A,'Z] THEN GOTO cctyes;
      =6: %LL%
         IF char IN ['a,'z] THEN GOTO cctyes;
      =2: %ULD%
         IF char IN ['A,'Z] OR
            char IN ['0,'9] THEN GOTO cctyes;
      =3: %LLD%
         IF char IN ['a,'z] OR
            char IN ['0,'9] THEN GOTO cctyes;
      ENDCASE;
   RETURN [FALSE] (char);
   (cctyes): RETURN [TRUE] (char)                                  12N3
   END.

(cct) PROCEDURE(chrcls);                                           120
   chrct(READC, chrcls);
   RETURN(mreg) END.

(cctf) PROCEDURE(chrcls);                                          12P
   WHILE (a1 _ READC) # ENDCHR DO
      BEGIN
      chrct(a1, chrcls);
      IF mreg THEN RETURN(TRUE);
      END;
   RETURN (FALSE) END.

(span) PROCEDURE(chrcls);                                          12Q
   LOCAL sw1;
   DO
      BEGIN
      sw1 _ swork1;
      chrct(READC, chrcls);
      END
      WHILE mreg;
   swork1 _ sw1;
   fxswork();
   RETURN END.

(tchf) PROCEDURE(char);                                            12R
   CASE READC OF
      =char: RETURN(TRUE);
      =ENDCHR: RETURN(FALSE);
      ENDCASE REPEAT CASE;
   END.

(cptse) PROCEDURE (stid);                                          12S
   %statement end t-pointer%
   %called by SPL compiler only%
```

```
    IF stid.stastr THEN
       BEGIN
       a2 _ [stid.stadr].L + 1;
       a1 _ stid;
       RETURN;
       END;
    flcpfse( stid );
    RETURN END.

% storage manipulation.  stacks, rings, buffers%
  DECLARE EXTERNAL FIELD
    systkp = [0(rp), 18:0], %pointer%
    systkt = [0(rp), 13:2], %type (1=stack, 2=ring)%
    systks = [1(rp), 18:18], %size of element%
    systke = [1(rp), 18:0]; %end of store%
    % rcpsh, rpop, rstsk, and <IDLIBE>gbotids use the fact that the
    stack body starts in word 2. %

  (rcpsh) %Record push.  Arguments are (1) address of record, and (2)
  address of stack or ring buffer.  The record is pushed on the
  store.%
                                                                  13B
    PROCEDURE (recadr, st);
    REF st;
    IF st.systkt NOT IN [1,2] THEN
       sysrcv(programbug,$"bad address in stack/ring manipulation",
       sysclr());
    IF (st.systkp _ st.systkp + st.systks) = st.systke THEN
       IF st.systkt = 1 THEN ABORT(stkoverflow,$"stack overflow")
       ELSE st.systkp _ &st + 2; %ring wrap around%
    IF st.systks = 1 THEN [st.systkp] _ [recadr]
    ELSE mvbfbf (recadr, st.systkp, st.systks);
    RETURN END.

  (rpop) %Record pop.  Argument is the address of a store.  Pop the
  top element.%
                                                                  13C
    PROCEDURE (st);
    REF st;
    IF st.systkt = 1 %stack% THEN
       IF st.systkp < &st+2 THEN ABORT(stkunderflow,$"stack
       underflow")
       ELSE NULL
    ELSE IF st.systkt = 2 %ring% THEN
       IF st.systkp <= &st+2 THEN
          st.systkp _ st.systke
       ELSE NULL
    ELSE sysrcv(programbug,$"bad address in stack/ring manipulation",
    sysclr());
    st.systkp _ st.systkp - st.systks;
    RETURN END.

  (rstsk) %Reset store%
                                                                  13D
    PROCEDURE (st);
```

```
    REF st;
    IF st.systkt NOT IN [1,2] THEN
       sysrcv(programbug,$"bad address in stack/ring manipulation",
       sysclr());
    st.systkp _ &st + 2 - st.systks;
    RETURN END.

(rcpto) %Record Pop To.  Arguments are (1) address of store, and (2)
address of record.  Pops the top record on the store to the
designated location.%
                                                                13E
    PROCEDURE (st, recadr);
    REF st;
    IF st.systkt NOT IN [1,2] THEN
       sysrcv(programbug,$"bad address in stack/ring manipulation",
       sysclr());
    IF st.systks = 1 THEN [recadr] _ [st.systkp]
    ELSE mvbfbf (st.systkp, recadr, st.systks);
    rpop (&st);
    RETURN END.

FINISH
```

(cbitst)	<nine, maconst, 061>	EXT CONSTANT =5 3A3E
(cblock)	<nine, maconst, 063>	EXT CONSTANT =8 3A3G
(cbooie)	<nine, maconst, 058>	EXT CONSTANT =2 3A3B
(cempty)	<nine, maconst, 057>	EXT CONSTANT =1 3A3A
(cindex)	<nine, maconst, 059>	EXT CONSTANT =3 3A3C
(cinteg)	<nine, maconst, 060>	EXT CONSTANT =4 3A3D
(clistt)	<nine, maconst, 064>	EXT CONSTANT =7 3A3H
(cstrin)	<nine, maconst, 062>	EXT CONSTANT =6 3A3F
(d1sel)	<nine, maconst, 072>	EXT CONSTANT =0 5A
(d2sel)	<nine, maconst, 0218>	EXT CONSTANT =2 5B
(ermalo)	<nine, maconst, 048>	EXT CONSTANT =15200B 4B1
(ermbadpcps1m)	<nine, maconst, 053>	EXT CONSTANT =15201B 4B2
(ermblksize)	<nine, maconst, 049>	EXT CONSTANT =15202B 4B3
(ermcnv)	<nine, maconst, 051>	EXT CONSTANT =15203B 4B4
(ermdesc)	<nine, maconst, 046>	EXT CONSTANT =15204B 4B5
(ermentity)	<nine, maconst, 039>	EXT CONSTANT =15205B 4B6
(ermhd)	<nine, maconst, 042>	EXT CONSTANT =15206B 4B7
(ermhov)	<nine, maconst, 041>	EXT CONSTANT =15207B 4B8
(ermhowpcp)	<nine, maconst, 068>	EXT CONSTANT =15210B 4B9
(ermhstor)	<nine, maconst, 043>	EXT CONSTANT =15211B 4B10
(ermillarg)	<nine, maconst, 038>	EXT CONSTANT =15212B 4B11
(erminit)	<nine, maconst, 066>	EXT CONSTANT =15213B 4B12
(ermmsg)	<nine, maconst, 0164>	EXT CONSTANT =15225B 4B22
(ermnargs)	<nine, maconst, 037>	EXT CONSTANT =15214B 4B13
(ermnoproc)	<nine, maconst, 036>	EXT CONSTANT =15215B 4B14
(ermnoroom)	<nine, maconst, 040>	EXT CONSTANT =15216B 4B15
(ermpkgtfull)	<nine, maconst, 065>	EXT CONSTANT =15217B 4B16
(ermpraddr)	<nine, maconst, 045>	EXT CONSTANT =15220B 4B17
(ermprname)	<nine, maconst, 044>	EXT CONSTANT =15221B 4B18
(ermptype)	<nine, maconst, 050>	EXT CONSTANT =15222B 4B19
(ermreturn)	<nine, maconst, 067>	EXT CONSTANT =15223B 4B20
(ermstring)	<nine, maconst, 047>	EXT CONSTANT =15224B 4B21
(ermunkown)	<nine, maconst, 073>	EXT CONSTANT =15226B 4B23
(lower)	<nine, maconst, 0214>	EXT CONSTANT =1 5C
(ubitst)	<nine, maconst, 030>	EXT CONSTANT =8 + lblock 3A2E
(ublock)	<nine, maconst, 031>	EXT CONSTANT =lblock 3A2H
(ubooie)	<nine, maconst, 028>	EXT CONSTANT =16 + linteg 3A2C
(uindex)	<nine, maconst, 027>	EXT CONSTANT =8 + linteg 3A2B
(uinteg)	<nine, maconst, 026>	EXT CONSTANT =linteg 3A2A
(ulist)	<nine, maconst, 033>	EXT CONSTANT =llist 3A2J
(unull)	<nine, maconst, 032>	EXT CONSTANT =lnull 3A2I
(upper)	<nine, maconst, 0215>	EXT CONSTANT =2 5D
(ustrin)	<nine, maconst, 029>	EXT CONSTANT =lstrin 3A2D
(utpblo)	<nine, maconst, 055>	EXT CONSTANT =24 + lblock 3A2G
(utpsbl)	<nine, maconst, 054>	EXT CONSTANT =16 + lblock 3A2F

```
< NINE, MACONST.NLS;2, >, 9-Jul-78 17:32 BLP ;;;;
FILE maconst % <arcsubsys,x110,> TO <relnine,maconst.rel,>%
%This file contains list constants, error codes and constants for the
MSG3, raw network or shared page versions of NLS-9.  The corresponding
file for the DPS (MAGIC) version of NLS-9 is <nine, mconst,>.%
%List constants%
   %List type definitions%
      %L10 List - The following branch should be in L10LRP%
         %DECLARE EXTERNAL CONSTANT  descriptor types %
            %
            lnull  = 0 ,   NULL
            linteg = 2 ,   INTEGER
            lstrin = 3 ,   STRING
            llist  = 4 ,   LIST
            lblock = 1 ,   BLOCK
            ldescr = 101 ,   DESCRIPTOR -- never stored as type
            lnewde = 100 ;   NEWDESCRIPTOR -- never stored as type
            %
      %L10 List types including user bits%
         (uinteg) EXTERNAL CONSTANT =  linteg; % 2 INTEGER %          3A2A
         (uindex) EXTERNAL CONSTANT = 8 + linteg; % 10 INDEX%        3A2B
         (uboole) EXTERNAL CONSTANT = 16 + linteg; % 18 BOOLEAN%     3A2C
         (ustrin) EXTERNAL CONSTANT = lstrin; % 3 STRING%            3A2D
         (ubitst) EXTERNAL CONSTANT = 8 + lblock; % 9 BITSTRING%    3A2E
         (utpsbl) EXTERNAL CONSTANT = 16 + lblock; % 17 block containg
         two text pointers and a string%                            3A2F
         (utpblo) EXTERNAL CONSTANT = 24 + lblock; % 25 block containg
         two text pointers%                                         3A2G
         (ublock) EXTERNAL CONSTANT = lblock; % 1 BLOCK%            3A2H
         (unull) EXTERNAL CONSTANT = lnull; % 0 NULL%               3A2I
         (ulist) EXTERNAL CONSTANT = llist; % 4 LIST%               3A2J
      %PCPB36 type codes%
         (cempty) EXTERNAL CONSTANT = 1; %EMPTY%                    3A3A
         (cboole) EXTERNAL CONSTANT = 2; %BOOLEAn %                 3A3B
         (cindex) EXTERNAL CONSTANT = 3; %INDEX%                    3A3C
         (cinteg) EXTERNAL CONSTANT = 4; %INTEGER%                  3A3D
         (cbitst) EXTERNAL CONSTANT =5; %BITSTRING%                 3A3E
         (cstrin) EXTERNAL CONSTANT = 6; %CHARACTER STRING%         3A3F
         (cblock) EXTERNAL CONSTANT = 8; %BLOCK%                    3A3G
         (clistt) EXTERNAL CONSTANT =7; %LIST%                      3A3H
%errors%                                                              4
   % Notification %
   % Program error %
      (ermalo) EXTERNAL CONSTANT = 15200B;                           4B1
      (ermbadpcpsim) EXTERNAL CONSTANT = 15201B;                     4B2
      (ermblksize) EXTERNAL CONSTANT =15202B;                        4B3
      (ermcnv) EXTERNAL CONSTANT = 15203B;                           4B4
      (ermdesc) EXTERNAL CONSTANT = 15204B;                          4B5
      (ermentity) EXTERNAL CONSTANT = 15205B;                        4B6
      (ermhd) EXTERNAL CONSTANT = 15206B;                            4B7
      (ermhov) EXTERNAL CONSTANT = 15207B;                           4B8
      (ermhowpcp) EXTERNAL CONSTANT = 15210B;                        4B9
      (ermhstor) EXTERNAL CONSTANT = 15211B;                        4B10
      (ermillarg) EXTERNAL CONSTANT = 15212B;                       4B11
      (erminit) EXTERNAL CONSTANT = 15213B;                         4B12
      (ermnargs) EXTERNAL CONSTANT = 15214B;                        4B13
```

```
    (ermnoproc) EXTERNAL CONSTANT = 15215B;                          4B14
    (ermnoroom) EXTERNAL CONSTANT = 15216B;                          4B15
    (ermpkgtfull) EXTERNAL CONSTANT = 15217B;                        4B16
    (ermpraddr) EXTERNAL CONSTANT = 15220B;                          4B17
    (ermprname) EXTERNAL CONSTANT = 15221B;                          4B18
    (ermptype) EXTERNAL CONSTANT = 15222B;                           4B19
    (ermreturn) EXTERNAL CONSTANT = 15223B;                          4B20
    (ermstring) EXTERNAL CONSTANT =15224B;                           4B21
    (ermmsg) EXTERNAL CONSTANT = 15225B;  %error interfacing with
    MSG3%
                                                                     4B22
    (ermunkown) EXTERNAL CONSTANT = 15226B;                          4B23
  % Fatal Error %
%misc%
  (d1sel) EXTERNAL CONSTANT = 0;                                      5A
  (d2sel) EXTERNAL CONSTANT = 2;                                      5B
  (lower) EXTERNAL CONSTANT = 1;                                      5C
  (upper) EXTERNAL CONSTANT = 2;                                      5D
FINISH
```