```
    %If &da is zero then restore the display image of all of the text
    display areas.  Otherwise, restore only the given text display
    area image.%
    %---------------%
    LOCAL  end;
    LOCAL LIST procpar[2], windatt[3];
    IF NOT &da THEN
       end _ (&da _ $dpyarea) + dacnt*dal
    ELSE
       end _ &da + dal;
    DO
       IF da.daexis AND NOT da.daseg AND da.dawid AND da.dasuppress
       AND NOT da.daauxiliary THEN
          BEGIN
          da.dasuppress _ FALSE;
          % assume commands list passed and that this will be
          executed by batch commands.  we should permit immediate
          execution; maybe permit &commands to be 0 in which case wee
          execute now? %
          #windatt# _
             USE makedesc( unull, 0, FALSE),
             USE makedesc( uindex, 1 %visible%, FALSE),
             USE makedesc( unull, 0, FALSE);
          #procpar# _
             USE makedesc( uindex, da.dawid, FALSE ), % window id %
             USE makedesc( ulist, $windatt, FALSE);
          addtobatch( &commands, $procpar, setwindatt );
          [findwa( da.dawid )].wiatt _ 1; % visible %
          END
    UNTIL (&da _ &da+dal) >= end;
    NULL-LISTS;
    RETURN;
    END.

FINISH of DISPLAY
(explct)   % CL:LB ;  explicit request for content checking %
PROCEDURE (charct, stid, oldls REF, oblock, da REF, sa REF, commands
REF LIST, staddr REF LIST);                                          10
  % Procedure description
    FUNCTION
       This routine processes a statement that is already on the
       screen and for which explicit content checking has been
       requested.  It uses stfrmt to format the statement.  If the
       statement is on a new position on the screen, it issues a
       replace statement command.  Otherwise it does a line segment
       by line segment check on the statement and issues a replace
       line segment commands for each line segment that was changed
       and write line segment commands if the new statement has more
       line segments than the old.
       This routine assumes that statement number/sids and signatures
       are line segments in the string containing the text of the
       statement and not separate strings.
    ARGUMENTS
       charct:  position in statement at which to begin display
       stid:  stid of statement to be replaced
       oldls:  address of first line segment old display of statement
```

```
         oblock:  address of block containing oldls
         da:  address of display area
         commands:  list of commands
         staddr:  list of string table entry addresses
      RESULTS
         none
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % Declarations %
      LOCAL festrngid, length, newls REF, nblock, moren, bp;
   %Format statement and set flags %
      stfrmt(&da, stmntid(&oldls), charct, sa.swclvl, sa.swslvl,
      sa.swsvw, sa.swvspec, sa.swvsp2, &commands, oldls.strngid:
      &newls, nblock);
      oldls.stold _ 0;
      festrngid _ oldls.strngid;
   %See if statement is at same position on screen%
      IF newls.sty NOT= oldls.sty THEN
         BEGIN  %new position.%
         DO  % build replace statement command%
            BEGIN  %call buildcom for each line segment%
            buildcom(&commands, &da, &newls, buildstring(newls.stbps,
            newls.stbpe), festrngid);
            END
         WHILE dnxtls(&newls, nblock, forward: &newls, nblock);
         DO  %flag old line segment as not needing to be deleted%
            oldls.stnew _ FALSE
         WHILE (dnxtls(&oldls, oblock: &oldls, oblock));
         END
      ELSE  %same position.  Use replace line sements%
         BEGIN
         %do line segment by line segment comparison between old
         display of statement and new display.  Build replace line
         segement command for any line segment that is either not at
         the same position or changed in content.  Copy FE string id
         and line segment id into new table %
            DO  %loop until out of line segments in old or new
            statement%
               BEGIN
               oldls.stnew _ FALSE;  %don't need delete command%
               newls.strngid _ festrngid;
               newls.stlsid _ oldls.stlsid;
               bp _
                  IF oldls.stsrce = srcstat THEN
                     chbptr(oldls.stccnt - 1) + ( IF stid = cdstd1 THEN
                     $cdstr1 ELSE $cdstr2 )
                  ELSE oldls.stbps;
               IF NOT ( newls.stx1 = oldls.stx1
               AND newls.sty = oldls.sty
               AND newls.stx2 = oldls.stx2
               AND (length _ slngth(newls.stbps, newls.stbpe)) =
               slngth(oldls.stbps, oldls.stbpe)
               AND complsg (newls.stbps, bp, length) ) THEN
```

```
                  bldrpl(&newls, &da, &commands);  %changed%
               END
            WHILE (moren _ dnxtls(&newls, nblock, forward: &newls,
            nblock)) AND dnxtls(&oldls, oblock: &oldls, oblock);
         %check for change in number of line segements and build write
         line segment commands as needed %
            IF moren THEN
               DO %more line segments in new%
                  BEGIN
                  newls.strngid _ festrngid;
                  bldwls(&newls, &da, &commands);
                  #staddr# !_ &newls;
                  END
               WHILE dnxtls(&newls, nblock, forward: &newls, nblock);
         END;
   % Return %
      RETURN;
   END.

% Obsolete procedures %
   (builldcom) PROCEDURE % appends the command to the commands list. %
                                                                  11A
      ( commands REF LIST,
      da REF,
      firstflag,
         % If TRUE, this is the first line segment in the string and
         the writestring command must be built up.  If FALSE, this is
         not the first line segment and the line segment should be
         appended to the list which is the third parameter of the last
         writestring command in the commands list, i.e., the fourth
         parameter of the last command appended to commands. %
      wid, % window id: ignored if firstflag is FALSE %
      x, % x coordinate of origin of line segment  %
      y, % y coordinate of origin of line segment-- BE internal
      coordinate system:  must be subtracted from da.dabottom to get FE
      y-coordinate.  (BE goes from 0 to da.dabottom; FE goes from
      da.dabottom  at top to 0 at bottom of window.) %
      attributes,
      selector,
      stringadr REF,
      stringid);
      LOCAL ptr REF;
      IF firstflag THEN
         BEGIN
         IF NOT stringid THEN
            BEGIN
            #commands# !_
               LIST(
                  USE makedesc( uindex, writstr, FALSE),
                  LIST( % parameters %
                     USE makedesc( uindex, wid, FALSE),
                     LIST( % string attributes %
                        LIST( x, da.dabottom-y),
                        USE makedesc( uindex, attributes, FALSE),
                        USE makedesc( uindex, selector, FALSE)
                        ),
                     LIST( % a list of line segments making up the
                     string %
                        LIST(
                           USE makedesc( unull, 0, FALSE), % use string
                           default as first line segment attributes %
                           *stringadr*
                           )
                        )
                     )
                  );
         END
      ELSE
         BEGIN
         #commands# !_
            LIST(
               USE makedesc( uindex, replstr, FALSE),
               LIST( % parameters %
                  USE makedesc( uindex, wid, FALSE),
                  USE makedesc( uindex, stringid, FALSE),
                  LIST( % string attributes %
                     LIST( x, da.dabottom-y),
                     USE makedesc( uindex, attributes, FALSE),
                     USE makedesc( uindex, selector, FALSE)
                     ),
                  LIST( % a list of line segments making up the
                  string %
                     LIST(
                        USE makedesc( unull, 0, FALSE), % use string
                        default as first line segment attributes %
                        *stringadr*
                        )
                     )
                  )
               );
         END;
      END
   ELSE
      BEGIN
      % Append the line segment to the list of line segments
      associated with this string. %
      &ptr _ ELEM #commands#[commands.L];
         % points to last element in commands list: the list
         indicating last command %
      &ptr _ ELEM #ptr#[2];
         % points to the second element in the last command list: a
         list of parameters to the write-string or replace-string
         procedure %
      &ptr _ ELEM #ptr#[IF stringid THEN 4 ELSE 3];
         % points to the last parameter: a list of line segments
         making up the string %
      % append the new line segment to the end os the line segment
      list of the last command. %
      #ptr# !_ LIST( % the line segment %
         LIST( % line segment attributes %
            LIST( x, da.dabottom-y),
            USE makedesc( uindex, attributes, FALSE),
            USE makedesc( uindex, selector, FALSE)
            ),
         *stringadr*
         );
      END;
   RETURN;
   END.
```

```
(bmark)          <nine, encapsulator, 019>          PROCEDURE      1D
(chgdbk)         <nine, encapsulator, 029>          PROCEDURE      1E
(crfork)         <nine, encapsulator, 039>          PROCEDURE      1F
(cthaw)          <nine, encapsulator, 061>          PROCEDURE      1G
(definetraps)    <nine, encapsulator, 066>          PROCEDURE      1H
(dtjsys)         <nine, encapsulator, 076>          PROCEDURE      1I
(dummyjsystrappsi) <nine, encapsulator, 095>        PROCEDURE      1J
(fkexist)        <nine, encapsulator, 0351>         PROCEDURE      1K
(fkstruc)        <nine, encapsulator, 0374>         RECORD         1B
(gethandler)     <nine, encapsulator, 0110>         PROCEDURE      1L
(gtfblk)         <nine, encapsulator, 0116>         PROCEDURE      1M
(gtfkfile)       <nine, encapsulator, 0393>         PROCEDURE      1N
(gtfstr)         <nine, encapsulator, 0145>         PROCEDURE      1O
(interior)       <nine, encapsulator, 0375>         FIELD - 18     1B1
(jsystrappsi)    <nine, encapsulator, 098>          LOCAL          1J3
(jtrace)         <nine, encapsulator, 0165>         PROCEDURE      1P
(killfork)       <nine, encapsulator, 0178>         PROCEDURE      1Q
(parallel)       <nine, encapsulator, 0376>         FIELD - 18     1B2
(pinfacs)        <nine, encapsulator, 0201>         PROCEDURE      1R
(ptfblk)         <nine, encapsulator, 0212>         PROCEDURE      1S
(ptfstr)         <nine, encapsulator, 0246>         PROCEDURE      1T
(rfhndl)         <nine, encapsulator, 0377>         FIELD - 18     1B3
(rinfacs)        <nine, encapsulator, 0403>         PROCEDURE      1U
(rstjtinterrupt) <nine, encapsulator, 0252>         PROCEDURE      1V
(setjtinterrupt) <nine, encapsulator, 0260>         PROCEDURE      1W
(settraps)       <nine, encapsulator, 0284>         PROCEDURE      1X
(skipreturn)     <nine, encapsulator, 0298>         PROCEDURE      1Y
(statwrd)        <nine, encapsulator, 0379>         FIELD - 36     1B5
(strtfk)         <nine, encapsulator, 0399>         PROCEDURE      1Z
(superior)       <nine, encapsulator, 0378>         FIELD - 18     1B4
(waitfork)       <nine, encapsulator, 0315>         PROCEDURE      1A@
(winfacs)        <nine, encapsulator, 0336>         PROCEDURE      1AA
(winfpc)         <nine, encapsulator, 0343>         PROCEDURE      1AB
```

```
< NINE, ENCAPSULATOR.NLS.7, >, 15-Oct-77 16:06 ROM ;;;;
FILE encapsulator % (arcsubsys, XL10,) (arcsubsys,l109,) to
(relnine,encapsulator.rel,) %
   ALLOW!
   (fkstruc) RECORD                                                        1B
      inferior[18],      % pointer to inferior fork structure %
      parallel[18],      % pointer to parallel fork structure %
      rfhndl[18],        % relative fork handle %
      superior[18],      % pointer to superior fork structure %
      statwrd[36];       % fork status word %
   EXTERNAL unwait;
   (bmark)       % set bit in bittable and set up dispatch table%
   PROCEDURE(jsysnum, handler);                                            1D
      LOCAL wordnum, mask, i, j;
      DIV (jsysnum)/36, wordnum, i;
      mask _ 1;
      FOR j _ 0 UP UNTIL  >= (35-i) DO mask _ mask*2;
      bittable[wordnum] _ bittable[wordnum] .V mask;
      DIV jsysnum/2, i, j;
      IF j THEN dtable[i].LH _ handler ELSE dtable[i].RH _ handler;
      RETURN;
      END.

   (chgdbk)      % change the debreak location %
   PROCEDURE( intlev, newadrs );                                           1F
      % This procedure will change the debreaking address. Users should
      be cautious and possibly put a JFCL in that location %
      LOCAL lvltab, dbkadrs;
      REF lvltab, dbkadrs;
      !rir( 485 );  % current fork %
      &lvltab _ R2.LH;
      &dbkadrs _  lvltab[intlev-1];  % level i = offset i-1 %
      dbkadrs _ newadrs;
      RETURN;
      END.

   (crfork)      % create fork & enable capabilitites %
   PROCEDURE;                                                              1F
      % Procedure description
         FUNCTION
            create fork & enable capabilitites
         ARGUMENTS
            none
         RESULTS
            A fork handle
         NON-STANDARD CONTROL
            none
         GLOBALS
            none
         %
      LOCAL fkhandl;
      % create fork %
         IF NOT SKIP !cfork( 2B11 ) THEN
            err($"cannot create new fork");
         fkhandl _ R1;
      % enable all capabilities %
      !epcap( fkhandl, -1, -1 );
   RETURN(fkhandl);
   END.

(cthaw)      % Thaw the frozen fork %
PROCEDURE( frkhandle );                                                1G
   R1 _ frkhandle;
   !JSYS 323B; % UTFRK  -- unfreeze him%
   RETURN;
   END.

(definetraps)         % define the JSYS to trap according to table %
PROCEDURE( trptbl );                                                   1H
   LOCAL i;
   REF trptbl;
   FOR i _ 0 UP 2 UNTIL >= 1023 DO
      BEGIN
      IF trptbl[i] = -1 THEN EXIT LOOP;
      bmark(trptbl[i],trptbl[i+1]);
      END;
   RETURN;
   END.

(dtjsys)     % dispatch a trapped jsys%
PROCEDURE;                                                             1I
   LOCAL proc, unfrz, frknum, infjsysn;
   REF proc;
   CASE TRUE OF
      = tops20flag: %DEC type trapping%
         BEGIN
            !JSYS 322B; % RTFRK %
            frknum _ R1;
            infjsysn _ R2.RH;
         END;
      # tops20flag: %tenex type trapping% % should check for =
      tenexflag %
         BEGIN
            IF NOT SKIP !JSYS 322B THEN err($"rtfrk error in
            dtjsys");
            infjsysn _ R1.RH;
            frknum _ R1.LH;
         END;
      ENDCASE
         err($"Trapping not implemented");
   !rtsts(frknum);
   infpc _ R2.RH;
   !rfacs(frknum,$infacs);
   IF gprint THEN jtrace( frknum, infjsysn, $infpc, $infacs );
   unfrz _ TRUE; %the default%
   IF &proc _ gethandler(infjsysn) THEN
      IF proc(frknum,infjsysn,$infpc,$infacs:unfrz) THEN
         BEGIN
         winfacs(frknum,$infacs);
         !sfork(frknum,infpc);
         END;
   RETURN(unfrz, frknum);
   END.

(dummyjsystrappsi)   %DO NOT CALL THE PROCEDURE, PSI ROUTINE%
PROCEDURE;                                                          1J
   LOCAL fkhndl;
   RETURN;
   (jsystrappsi):                                                   1J3
   % save the accumulators %
      encsvl _ R1; R1 _ $encsvacs; !BLT R1, encsvend;
   % handle the JSYS %
      IF dtjsys( : fkhndl ) THEN cthaw( fkhndl );
   % restore the accumulators %
      !HRLZI R1, encsvacs;
      !BLT R1, 17B;
      R1 _ encsvl;
   % go back where you belong %
      !JSYS debrk;
   END.

(tkexist)    % check if fork exists %
PROCEDURE( frkh );                                                  1K
   % Procedure description
      FUNCTION
         check if fork exists
      ARGUMENTS
         fkhndl -- a fork handle
      RESULTS
         TRUE if exists FALSE if doesn"t
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   LOCAL ac, fork REF, struc[99];
   IF frkh = 0 THEN RETURN(FALSE);
   &fork _ $struc;
   IF tops20flag THEN
      BEGIN
         ac.LH _ -99;  ac.RH _ &fork;
         IF NOT SKIP !gfrks( 4B5, 4B11, ac) THEN RETURN(FALSE);
      END
   ELSE %tenex %
      BEGIN
         !gfrks( 4B5, 4B11+&fork );
      END;
   IF NOT &fork _ fork.inferior THEN RETURN(FALSE);
   DO % search structure for frkh %
      IF fork.rfhndl = frkh THEN RETURN(TRUE)
   UNTIL NOT &fork _ fork.parallel;
   RETURN(FALSE);
   END.

(gethandler) % get dispatch address for jsys handler%
PROCEDURE(n);                                                       1L
   LOCAL i, j, addr;
   DIV n/2, i, j;
   IF j THEN addr _ dtable[i].LH ELSE addr _ dtable[i].RH;
   RETURN(addr);
   END.

(gtfblk)     % get block from fork's address space %
PROCEDURE (locadrs, fkhndl, frkadrs, howmany);                      1M
   LOCAL count, frkpage, mypage, offset;
   REF locadrs, frkadrs ;
   mypage _ $ckpag/1000B;
   count _ 0;                                                       1M4
   &frkadrs.LH _ 0;
   WHILE (howmany _ howmany - count) > 0 DO                         1M6
      BEGIN                                                        1M6A
         % seperate page number and offset %
            DIV &frkadrs/1000B, frkpage, offset;
         % check for page overflow %
            count _ IF offset + howmany > 512 THEN 512-offset ELSE
            howmany;                                             1M6A2A
         % map the page %
            R1.LH _ fkhndl;
            R1.RH _ frkpage;
            R2.LH _ 400000B;  % current fork %
            R2.RH _ mypage;
            R3 _ 1B11; %read access only%
            !pmap();
         % move from mypage to locadress %
            mvbfbf( mypage*1000B + offset, &locadrs, count );  1M6A4A
            IF &frkadrs < 16 THEN % copy accumulators %        1M6A4B
               mvbfbf( $infacs+&frkadrs, &locadrs, 16-&frkadrs );
                                                                1M6A4B1
         % increment adresses %
            &frkadrs _ &frkadrs + count;                          1M6A5A
            &locadrs _ &locadrs + count;                          1M6A5B
      END;                                                         1M6B
   RETURN;
   END.

(gtfkfile)   % get a file into a fork %
PROCEDURE( fkhndl, jfn );                                           1N
   R1.LH _ fkhndl;
   R1.RH _ jfn;
   !get();
   RETURN;
   END.

(gtfstr)     % get ASCIZ string from fork's address space %
PROCEDURE(astr, fkhndl, fkbptr );                                   1O
   LOCAL i ;                                                       1O1
   REF astr;                                                       1O2
   gtfblk( &astr+1, fkhndl, fkbptr, (astr.M+4)/5 );                1O3
   IF fkbptr.LH = 777777B THEN fkbptr.LH _ 440700B;  %default case%
   R1.LH _ fkbptr.LH; %byte pointer %                              1O5
   R1.RH _ &astr+1;                                                1O6
   R2 _ 0;                                                         1O7
   FOR i _ 0 UP UNTIL > astr.M DO                                  1O8
      BEGIN                                                       1O8A
```

```
         !ILDB R2,R1;
         IF R2 = 0 THEN EXIT LOOP;                                    108A2
      END;                                                            108B
   astr.L _ i;                                                         109
   CASE (i _ 5 - fkbptr.LH/7B4) OF
      IN [1 , 5]: *astr*[1 TO i] _ NULL;
      = 0: NULL;
      ENDCASE dismes(1, $"illegal byte increment");
   RETURN;                                                            1011
   END.
                                                                      1012
(jtrace)     % Print trace info to tty: %
PROCEDURE( frkn, jnum, infaddr, regs);                                  1P
   REF infaddr, regs, jnum;
   LOCAL STRING work[50];
      *work* _ "Trapping fork ", STRING((frkn - 400000B),8), " JSYS
      ";
      IF &jnum < 512 THEN
         *work* _ *work*, "# ", STRING(&jnum,8)
      ELSE
         *work* _ *work* , *jnum*;
      *work* _ EOL, *work*,  "  AT ",STRING(infaddr,8), EOL ;
      dismes(1, $work);
      pinfacs(&regs);
      RETURN;
      END.

(killfork)   % kill the fork %
PROCEDURE( fkhndl );                                                    1Q
   % Procedure description
      FUNCTION
         kill the specified fork if it exists
      ARGUMENTS
         fkhndl -- a fork handle
      RESULTS
         none
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % do nothing if fork doesn't exist %
      IF NOT fkexist( fkhndl ) THEN RETURN;
   % make sure it is frozen %
      !ffork( fkhndl );
   % kill it %
      !kfork( fkhndl );
   RETURN;
   END.

(pinfacs)    % print contents of inferior forks regs %
PROCEDURE;                                                              1R
   LOCAL i;
   LOCAL STRING f[100];
   FOR i _ 1 UP 1 UNTIL > 4 DO
      BEGIN
```

```
        *f* _ "ACC", STRING(i), "] = ", STRING(infacs[i].LH,8), ",,",
        STRING(infacs[i].RH,8) ;
        dismes(1, $f);
        crlf();
        END;
     RETURN;
     END.

(pttblk)     % write a block into fork's address space %
PROCEDURE (locadrs, fkhndl, frkadrs, howmany);                          1S
     LOCAL count, frkpage, mypage, offset;
     REF locadrs, frkadrs ;
     mypage _ $ckpag/1000B;
     count _ 0;                                                         1S4
     &frkadrs.LH _ 0;
     IF &frkadrs < 20B THEN % Write in ACS %
        BEGIN
           count _ MIN( howmany, 20B-&frkadrs);
           mvbfbf( &locadrs, $infacs+&frkadrs, count );
           winfacs( fkhndl, $infacs );
           &frkadrs _ 20B;
        END;
     WHILE (howmany _ howmany - count) > 0 DO                           1S7
        BEGIN                                                          1S7A
           % seperate page number and offset %
              DIV &frkadrs/1000B, frkpage, offset;
           % check for page overflow %
              count _ IF offset + howmany > 512 THEN 512-offset ELSE
              howmany;                                               1S7A2A
           % map the page %
              R1.LH _ fkhndl;
              R1.RH _ frkpage;
              R2.LH _ 400000B;  % current fork %
              R2.RH _ mypage;
              R3 _ 14B10; % read/write access%
              !pmap();
           % move from mypage to locadress %
              mvbfbf( &locadrs, mypage*1000B + offset, count );      1S7A4A
           % increment adresses %
              &frkadrs _ &frkadrs + count;                           1S7A5A
              &locadrs _ &locadrs + count;                           1S7A5B
        END;                                                           1S7B
     RETURN;
     END.

(ptfstr)     % write ASCIZ string into fork's address space %
PROCEDURE(astr, fkhndl, frkadrs );                                      1T
     LOCAL i ;                                                          1T1
     REF astr;                                                          1T2
     gtfblk( &astr+1, fkhndl, frkadrs, (astr.M+4)/5 );                  1T3
     RETURN;                                                            1T4
     END.
                                                                        1T5
(rinfacs)     % read inferior fork ACs %
PROCEDURE(fkhndl, acs REF);                                             1U
     !rfacs( fkhndl, &acs );
```

```
   RETURN;
   END.

(rstjtinterrupt)     % reset interrupt for jsys handling %
PROCEDURE( chan );                                                  1V
   % deactivates channel %
   R2 _ 1;
   R1 _ 35 - (chan MOD 36 );
   !LSH R2,(R1);
   !dic( 4B5 );
   RETURN;
   END.

(setjtinterrupt)  % set interrupt for jsys trapping %
PROCEDURE(chan, intlev);                                            1W
   LOCAL i, chntab, lvltab; REF  chntab, lvltab;
      R1 _ 400000B;
      !JSYS rir;
      &lvltab _ R2.LH;
      &chntab _ R2.RH;
      i.LH _ intlev MOD 4;
      i.RH _ 0;
      chntab[chan] _ $jsystrappsi .V i; % jsys trap psi routine,
      level intlev%
      R1 _ 4B5;
      R2 _ &chntab;
      R2.LH _ &lvltab;
      !JSYS sir;
   %enable interrupts%
      R1 _ 4B5;
      !JSYS eir;
   %activate jsys trap pseudo interrupt%
      R2 _ 1;
      R1 _ 35 - (chan MOD 36 );
      !LSH R2,(R1);
      R1 _ 4B5;
      !JSYS aic;
   RETURN;
   END.

(settraps)   % set trapping of inferior JSYS %
PROCEDURE( fkhndl, chan, intlev );                                  1X
   !ffork( fkhndl ); % make sure it's frozen %
   setjtinterrupt( chan, intlev );
   CASE TRUE OF
      = tops20flag: %DEC type trapping%
         BEGIN
            R1.LH _ 0;  % set the traps %
            R1.RH _ fkhndl;
            R2.LH _ chan;
            R2.RH _ 1000B;
            R3    _ $bittable;
            !JSYS 321B;  % TFORK %
            R1.LH _ 3;  % set PSI channel %
            R1.RH _ fkhndl;
            R2.LH _ chan;
            R2.RH _ 1000B;
            R3    _ $bittable;
            !JSYS 321B;  % TFORK %
         END;
      # tops20flag: %tenex type trapping% % should check for =
      tenexflag %
         BEGIN
            R1.LH _ 4B4;
            R1.RH _ fkhndl;
            R2.LH _ chan;
            R2.RH _ $bittable;
            IF NOT SKIP !JSYS 321B THEN err($"tfork error in setting
            channel #");
            R1.LH _ 4B5;
            R1.RH _ fkhndl;
            R2.LH _ chan;
            R2.RH _ $bittable;
            IF NOT SKIP !JSYS 321B THEN err($"tfork error in setting
            traps");
         END;
      ENDCASE
         err($"Trapping not implemented");
   !rfork( fkhndl ); % unfreeze it %
   RETURN;
   END.

(skipreturn) % bump infpc for skip return %
PROCEDURE( frkpc );                                                 1Y
   % Procedure description
      FUNCTION
         bump infpc for skip return
      ARGUMENTS
         frkpc -- a pointer to where the encapsulator holds the
         fork's PC
      RESULTS
         none
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   REF frkpc;
   BUMP frkpc;
   RETURN;
   END.

(strtfk)             % start a fork at entry vector %
PROCEDURE( fkhndl, evctor );                                        1Z
   !strkv( fkhndl, evctor );
   RETURN;
   END.

(waitfork)           % universal waiting location %
PROCEDURE( fkhndl );                                                1A@
   % Procedure description
      FUNCTION
```

```
         This procedure will wait until a fork terminates and then
         kill it.  A retun label is provided so that the killing of
         the fork can be skipped.
      ARGUMENTS
         fkhndl -- fork handle to wait for termination
      RESULTS
         none
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % wait for fork termination %
      !wfork( fkhndl );
   % kill the fork %
      killfork( fkhndl );
   % come to here if you don't want the fork killed %
      (unwait): !JFCL 0;                                         1A@4A
   RETURN;
   END.

(winfacs)    % change inferior forks AC's%
PROCEDURE( fkhndl, blkaddr);                                     1AA
   REF blkaddr;
   R1 _ fkhndl;
   R2 _ &blkaddr;
   !sfacs();
   RETURN;
   END.

(winfpc)     % change inferior forks PC%
PROCEDURE( fkhndl, newpc);                                       1AB
   R1 _ fkhndl;
   R2 _ newpc;
   !sfork();
   RETURN;
   END.

FINISH of encapsulator
```

```
(badappend)     <nine, error, 089>            EXT CONSTANT =14110B 6A9
(badarg)        <nine, error, 071>            EXT CONSTANT =14100B 6A1
(badbufsize)    <nine, error, 0108>           EXT CONSTANT =15400B 9B1
(badcase)       <nine, error, 093>            EXT CONSTANT =14111B 6A10
(badcreate)     <nine, error, 088>            EXT CONSTANT =14112B 6A11
(badda)         <nine, error, 0109>           EXT CONSTANT =16301B 8C2
(baddate)       <nine, error, 094>            EXT CONSTANT =14113B 6A12
(baddelete)     <nine, error, 091>            EXT CONSTANT =14114B 6A13
(badedel)       <nine, error, 090>            EXT CONSTANT =14115B 6A14
(bademove)      <nine, error, 098>            EXT CONSTANT =14116B 6A15
(badfname)      <nine, error, 0101>           EXT CONSTANT =14001B 5A1
(badgrp)        <nine, error, 0100>           EXT CONSTANT =14101B 6A2
(badinstruction) <nine, error, 082>           EXT CONSTANT =16700B 12C1
(badload)       <nine, error, 0218>           EXT CONSTANT =14406B 9A7
(badmerge)      <nine, error, 097>            EXT CONSTANT =14117B 6A16
(badmove)       <nine, error, 099>            EXT CONSTANT =14120B 6A17
(badnum)        <nine, error, 0118>           EXT CONSTANT =14501B 10A1
(badsort)       <nine, error, 0110>           EXT CONSTANT =14121B 6A18
(badsplit)      <nine, error, 095>            EXT CONSTANT =14122B 6A19
(badstid)       <nine, error, 0113>           EXT CONSTANT =14123B 6A20
(badstmt)       <nine, error, 096>            EXT CONSTANT =14102B 6A3
(badsub)        <nine, error, 0111>           EXT CONSTANT =14124B 6A21
(badtranspose)  <nine, error, 0112>           EXT CONSTANT =14125B 6A22
(badupdate)     <nine, error, 085>            EXT CONSTANT =14126B 6A23
(cchangestring) <nine, error, 0183>           EXT CONSTANT =14400B 9A1
(cgothelp)      <nine, error, 0181>           EXT CONSTANT =14401B 9A2
(clisterr)      <nine, error, 0213>           EXT CONSTANT =15702B 12B2
(clistspace)    <nine, error, 0188>           EXT CONSTANT =16400B 9C1
(cnohelp)       <nine, error, 0178>           EXT CONSTANT =14402B 9A3
(conto)         <nine, error, 0220>           EXT CONSTANT =15304B 8B4
(cprogrambug)   <nine, error, 0189>           EXT CONSTANT =16401B 9C2
(creturn)       <nine, error, 0179>           EXT CONSTANT =14403B 9A4
(csaroverflow)  <nine, error, 0184>           EXT CONSTANT =15401B 9B2
(cstkoverflow)  <nine, error, 0185>           EXT CONSTANT =16402B 9C3
(cstkunderflow) <nine, error, 0186>           EXT CONSTANT =16403B 9C4
(cuncaughtabort) <nine, error, 0187>          EXT CONSTANT =16404B 9C5
(cunwind)       <nine, error, 0180>           EXT CONSTANT =14405B 9A6
(eof)           <nine, error, 0104>           EXT CONSTANT =14005B 5A5
(erdfedisplay)  <nine, error, 0193>           EXT CONSTANT =16300B 8C1
(erdprogram)    <nine, error, 0194>           EXT CONSTANT =15301B 8B1
(erdwindow)     <nine, error, 0195>           EXT CONSTANT =14300B 8A1
(erfeshow)      <nine, error, 0216>           EXT CONSTANT =14302B 8A3
(errsig)        <nine, error, 079>            EXT CONSTANT =14704B 12A5
(filebad)       <nine, error, 0106>           EXT CONSTANT =14003B 5A3
(filenotfound)  <nine, error, 0105>           EXT CONSTANT =14002B 5A2
(frremptyentry) <nine, error, 0215>           EXT CONSTANT =14705B 12A6
(ftpsig)        <nine, error, 081>            EXT CONSTANT =15701B 12B1
(gaderr)        <nine, error, 078>            EXT CONSTANT =14520B 10A11
(ink2err)       <nine, error, 053>            EXT CONSTANT =14503B 10A3
(ink5err)       <nine, error, 059>            EXT CONSTANT =14506B 10A6
(ink6err)       <nine, error, 061>            EXT CONSTANT =14507B 10A7
(ink7err)       <nine, error, 063>            EXT CONSTANT =14510B 10A8
(ink9err)       <nine, error, 067>            EXT CONSTANT =14512B 10A10
(longmarker)    <nine, error, 092>            EXT CONSTANT =14103B 6A4
(nobtmn)        <nine, error, 0115>           EXT CONSTANT =15302B 8B2
(nojrnl)        <nine, error, 0102>           EXT CONSTANT =14600B 11A1
```

```
(noiertn)        <nine, error, 0117>              EXT CONSTANT =15303B 8B3
(nomail)         <nine, error, 0103>              EXT CONSTANT =14601B 11A2
(nomod)          <nine, error, 086>               EXT CONSTANT =14104B 6A5
(noremote)       <nine, error, 087>               EXT CONSTANT =14105B 6A6
(nortn)          <nine, error, 0116>              EXT CONSTANT =15001B 5B1
(notopn)         <nine, error, 0114>              EXT CONSTANT =14004B 5A4
(notyet)         <nine, error, 0190>              EXT CONSTANT =14700B 12A1
(nowmod)         <nine, error, 083>               EXT CONSTANT =14106B 6A7
(nowrtacc)       <nine, error, 0209>              EXT CONSTANT =14007B 5A7
(ofilerr)        <nine, error, 076>               EXT CONSTANT =14006B 5A6
(prcerr)         <nine, error, 077>               EXT CONSTANT =14602B 11A3
(repeatsearch)   <nine, error, 0107>              EXT CONSTANT =14107B 6A8
(smallda)        <nine, error, 084>               EXT CONSTANT =14301B 8A2
(sorterr)        <nine, error, 074>               EXT CONSTANT =15602B 11B2
(spacerr)        <nine, error, 075>               EXT CONSTANT =15402B 9B3
(sgerr)          <nine, error, 072>               EXT CONSTANT =15403B 9B4
(statesig)       <nine, error, 080>               EXT CONSTANT =14701B 12A2
(unknownerr)     <nine, error, 0196>              EXT CONSTANT =14702B 12A3
(upgerr)         <nine, error, 073>               EXT CONSTANT =15601B 11B1
(userterminate) <nine, error, 0207>               EXT CONSTANT =14703B 12A4
(werrsig)        <nine, error, 0191>              EXT CONSTANT =16701B 12C2
(wrttat)         <nine, error, 0210>              EXT CONSTANT =16001B 5C1
(zstringoverflow) <nine, error, 0182>             EXT CONSTANT =14404B 9A5
```

```
< NINE, ERROR.NLS;5, >, 1-Jun-78 12:06 HGL ;;;;
FILE error  % <ARCSUBSYS>XL10  <RELNINE>error %   % (arcsubsys,xL10,)
(arcsubsys,l109,) (RELNINE,error.rel,) %
% This file contains ALL error codes for the NLS BE.  They are arranged
in classes according to the scheme proposed in  (28074,).
   The numbers 14nnnB refer to NOTIFICATIONS.
   The numbers 15nnnB refer to NON-FATAL PROGRAM ERRORS.
   The numbers 16nnnB refer to FATAL ERRORS.
THERE SHOULD BE NO OTHER ERROR CODES DECLARED IN THE NLS BE! %
% Additionally, the Middle End converts the L10 system errors declared
in the file (nls, l10syms,) to the standard code for NLS BE errors.
The internal codes and their converted equivalents are listed here:
   helptype=1,   HELP
   notetype=2,   NOTE
   aborttype=3;  ABORT
   nohelp=10000B,   on resume, means no help obtained: cnohelp=14402B
        happens when HELP is not caught
   return=10001B,   a NOTE: procedure returning: creturn=14403B
        Whenever a procedure or its coroutine did an INVOKE, and then
      does RETURN
   unwind=10002B,   NOTE, this routine will vanish: cunwind=14405B
        happens when higher routine does TERMINATE
        also when runtime package does a recover
   gothelp=10003B,   on resume, means help obtained: cgothelp=14401B
        Should be standard first argument for RESUME
   stringoverflow=10004B,   HELP, string overflowed:
   zstringoverflow=14404B
        low level string handlers generate this
        arg2 is string address
        they expect resume(gothelp,new-string-address)
        IF program not able to provide new string, treat as ABORT
   changestring=10005B,   NOTE, arg2=old addr, arg3=new addr:
   cchangestring=14400B
        NOTE issued by low level string handlers after they got a new
      string address
   saroverflow=10006B,   ABORT, SAR string overflow:
   csaroverflow=15401B
        generated by runtime routines when SAR overflows
        perhaps later treated like other strings
   stkoverflow=10007B,   ABORT, program defined stack overflow:
   cstkoverflow= 16402B
   stkunderflow=10010B,   ABORT, program defined stack underflow:
   cstkunderflow=16403B
   listspace=10011B;   ABORT, list allocation zone full:
   clistspace=16400B
   programbug=20001B,   a program bug involved: cprogrambug=16401B
   uncaughtabort=20002B;   uncaught ABORT: cuncaughtabort=16404B
   %
% File system: N = 001B - 077B %
   % Notification %
      (badfname)          EXTERNAL CONSTANT = 14001B;                 5A1
      (filenotfound)      EXTERNAL CONSTANT = 14002B;                 5A2
      (filebad) EXTERNAL CONSTANT = 14003B;                           5A3
      (notopn)            EXTERNAL CONSTANT = 14004B;                 5A4
      (eof)               EXTERNAL CONSTANT = 14005B;                 5A5
      (ofilerr) EXTERNAL CONSTANT = 14006B;                           5A6
```

```
        % can't open a file %
      (nowrtacc)          EXTERNAL CONSTANT = 14007B;                        5A7
   % Program error %
      (nortn)             EXTERNAL CONSTANT = 15001B;                        5B1
   % Fatal error %
      (wrtfat)            EXTERNAL CONSTANT = 16001B;                        5C1
% Execution functions: N = 100B - 177B %
   % Notification %
      (badarg)            EXTERNAL CONSTANT = 14100B;                        6A1
      (badgrp)            EXTERNAL CONSTANT = 14101B;                        6A2
      (badstmt)           EXTERNAL CONSTANT = 14102B;                        6A3
      (longmarker)        EXTERNAL CONSTANT = 14103B;                        6A4
      (nomod)             EXTERNAL CONSTANT = 14104B;                        6A5
      (noremote)          EXTERNAL CONSTANT = 14105B;                        6A6
      (nowmod)            EXTERNAL CONSTANT = 14106B;                        6A7
      (repeatsearch)      EXTERNAL CONSTANT = 14107B;                        6A8
      (badappend)         EXTERNAL CONSTANT = 14110B;                        6A9
      (badcase)           EXTERNAL CONSTANT = 14111B;                       6A10
      (badcreate)         EXTERNAL CONSTANT = 14112B;                       6A11
      (baddate)           EXTERNAL CONSTANT = 14113B;                       6A12
      (baddelete)         EXTERNAL CONSTANT = 14114B;                       6A13
      (badedel)           EXTERNAL CONSTANT = 14115B;                       6A14
      (bademove)          EXTERNAL CONSTANT = 14116B;                       6A15
      (badmerge)          EXTERNAL CONSTANT = 14117B;                       6A16
      (badmove)           EXTERNAL CONSTANT = 14120B;                       6A17
      (badsort)           EXTERNAL CONSTANT = 14121B;                       6A18
      (badsplit)          EXTERNAL CONSTANT = 14122B;                       6A19
      (badstid)           EXTERNAL CONSTANT = 14123B;                       6A20
      (badsub)            EXTERNAL CONSTANT = 14124B;                       6A21
      (badtranspose)      EXTERNAL CONSTANT = 14125B;                       6A22
      (badupdate)         EXTERNAL CONSTANT = 14126B;                       6A23
   % Program error %
   % Fatal error %
% Middle end: N = 200B - 277B %
   % These are declared in (nine, mconst, errors) %
% Display code and formatters: N = 300B - 377B %
   % Notification %
      (erdwindow)         EXTERNAL CONSTANT = 14300B;                        8A1
         %error in split screen routine - recoverable%
      (smallda)           EXTERNAL CONSTANT = 14301B;                        8A2
      (erfeshow)                  EXTERNAL CONSTANT = 14302B;                8A3
         % user typed <CD> at scroll %
   % Program error %
      (erdprogram)        EXTERNAL CONSTANT = 15301B;  %program error% 8B1
      (nobtmn)            EXTERNAL CONSTANT = 15302B;                        8B2
      (noleftn)           EXTERNAL CONSTANT = 15303B;                        8B3
      (conto)             EXTERNAL CONSTANT = 15304B; % control-O reset
      failure in FE %                                                        8B4
   % Fatal error %
      (erdfedisplay)      EXTERNAL CONSTANT = 16300B;                        8C1
         %error in FE display package procedure%
      (badda)             EXTERNAL CONSTANT = 16301B;                        8C2
% Utility code: storage management, string construction, loading
programs: N = 400B - 477B %
   % Notification %
      (cchangestring)     EXTERNAL CONSTANT = 14400B;                        9A1
```

```
            % changestring=10005B;   NOTE, arg2=old addr, arg3=new addr:
            NOTE issued by low level string handlers after they got a new
            string address %
         (cgothelp)           EXTERNAL CONSTANT = 14401B;                9A2
            % gothelp=10003B;   on resume, means help obtained:   Should
            be standard first argument for RESUME %
         (cnohelp)            EXTERNAL CONSTANT = 14402B;                9A3
            % nohelp=10000B;   on resume, means no help obtained:  happens
            when HELP is not caught %
         (creturn)            EXTERNAL CONSTANT = 14403B;                9A4
            % return=10001B;   a NOTE: procedure returning:  Whenever a
            procedure or its coroutine did an INVOKE, and then RETURN %
         (zstringoverflow) EXTERNAL CONSTANT = 14404B;                   9A5
            % stringoverflow=10004B;   HELP, string overflowed:   low
            level string handlers generate this arg2 is string address
            they expect resume(gothelp,new-string-address).  IF program
            not able to provide new string, treat as ABORT %
         (cunwind)            EXTERNAL CONSTANT = 14405B;                9A6
            % unwind=10002B;   NOTE, this routine will vanish:   happens
            when higher routine does TERMINATE also when runtime package
            does a recover %
         (badload) EXTERNAL CONSTANT = 14406B;                           9A7
            %subsystem load failed%
      % Program error %
         (badbufsize)         EXTERNAL CONSTANT = 15400B;                9B1
         (csaroverflow)       EXTERNAL CONSTANT = 15401B;                9B2
            % saroverflow=10006B;   ABORT, SAR string overflow: %
         (spacerr)            EXTERNAL CONSTANT = 15402B;                9B3
            % display support didn't get space%
         (sqerr)              EXTERNAL CONSTANT = 15403B;                9B4
            % sequence generator error code %
      % Fatal error %
         (clistspace)         EXTERNAL CONSTANT = 16400B;                9C1
            % listspace=10011B;   ABORT, list allocation zone full: %
         (cprogrambug)        EXTERNAL CONSTANT = 16401B;                9C2
            % programbug=20001B;   a program bug involved: %
         (cstkoverflow)       EXTERNAL CONSTANT = 16402B;                9C3
            % stkoverflow=10007B;   ABORT, program defined stack
            overflow:%
         (cstkunderflow)      EXTERNAL CONSTANT = 16403B;                9C4
            % stkunderflow=10010B;   ABORT, program defined stack
            underflow: %
         (cuncaughtabort)     EXTERNAL CONSTANT = 16404B;                9C5
            % uncaughtabort=20002B;   uncaught ABORT: %
   % Addressing errors: N = 500B - 577B %
      % Notification %
         (badnum)             EXTERNAL CONSTANT = 14501B;               10A1
         % (lnk1err)          EXTERNAL CONSTANT = 14502B;
            Illegal Link Syntax:
            ENDCHR Before Closing " %
         (lnk2err)            EXTERNAL CONSTANT = 14503B;               10A3
            % Illegal Link Syntax:
            ENDCHR Before CH After ' %
         % (lnk3err)          EXTERNAL CONSTANT = 14504B;
            Illegal Link Syntax:
            Illegal Statement Name or Number After ! or * %
      % (lnk4err)        EXTERNAL CONSTANT = 14505B;
         Illegal Link Syntax:
         Illegal DAE Element %
      (lnk5err)          EXTERNAL CONSTANT = 14506B;                 10A6
         % Illegal Link:
         Left Delimeter Not Found %
      (lnk6err)          EXTERNAL CONSTANT = 14507B;                 10A7
         % Illegal Link Syntax or Semantic:
         Missing Right Delimeter or Bad Viewspecs %
      (lnk7err)          EXTERNAL CONSTANT = 14510B;                 10A8
         % Illegal Link Syntax:
         Missing Right Delimeter %
      % (lnk8err)        EXTERNAL CONSTANT = 14511B;
         Illegal Link Syntax:
         Illegal Marker After # %
      (lnk9err)          EXTERNAL CONSTANT = 14512B;                 10A10
         % Illegal Link Syntax:
         ENDCHR Before ; In Filter %
      (gaderr)           EXTERNAL CONSTANT = 14520B;  %content/word
      search error%                                                  10A11
   % Program error %
   % Fatal error %
% Subsystems: N = 600B - 677B %
   % Notification %
      (nojrnl)           EXTERNAL CONSTANT = 14600B;                 11A1
      (nomail)           EXTERNAL CONSTANT = 14601B;                 11A2
      (prcerr)           EXTERNAL CONSTANT = 14602B;                 11A3
         % can't get a jfn for a processor %
   % Program error %
      (upgerr)           EXTERNAL CONSTANT = 15601B;                 11B1
         % user program error code %
      (sorterr)          EXTERNAL CONSTANT = 15602B;                 11B2
         % sort-merge error %
   % Fatal Error %
% Miscellaneous: N = 700B - 777B %
   % Notification %
      (notyet)           EXTERNAL CONSTANT = 14700B;                 12A1
      (statesig)         EXTERNAL CONSTANT = 14701B;                 12A2
      (unknownerr)       EXTERNAL CONSTANT = 14702B;                 12A3
      (userterminate)    EXTERNAL CONSTANT = 14703B;                 12A4
         %user terminated command%
      (errsig)           EXTERNAL CONSTANT = 14704B;                 12A5
         %code for ABORT in procedure 'err'%
      (trremptyentry)    EXTERNAL CONSTANT = 14705B;                 12A6
   % Program error %
      (ftpsig)           EXTERNAL CONSTANT = 15701B;                 12B1
      (clisterr)         EXTERNAL CONSTANT = 15702B;                 12B2
         %corres. list out of space%
   % Fatal Error %
      (badinstruction)   EXTERNAL CONSTANT = 16700B;                 12C1
         %illegal (instruction%
      (werrsig)          EXTERNAL CONSTANT = 16701B;    % ? %        12C2
FINISH
```

```
(acisnum)      <nine, execfl, 02111>      LOCAL          5N10
(acisstr)      <nine, execfl, 02089>      LOCAL          5N9
(arcflandpc)   <nine, execfl, 0741>       LOCAL          4K
(blksiz)       <nine, execfl, 01196>      LOCAL          5B1A
(cflapc)       <nine, execfl, 0929>       LOCAL          4O
(chkdev)       <nine, execfl, 0353>       LOCAL          4C
(chkpcs)       <nine, execfl, 0368>       LOCAL          4D
(copflandpc)   <nine, execfl, 0882>       LOCAL          4M
(delflandpc)   <nine, execfl, 0468>       LOCAL          4G
(dlbgac)       <nine, execfl, 02069>      LOCAL          5N
(dlbgas)       <nine, execfl, 02219>      LOCAL          5R
(dlbglw)       <nine, execfl, 02184>      LOCAL          5Q
(dlbgpr)       <nine, execfl, 02150>      LOCAL          5P
(dlbgtd)       <nine, execfl, 02117>      LOCAL          5O
(dlbidai)      <nine, execfl, 01354>      LOCAL          5F
(dlbidi)       <nine, execfl, 01401>      LOCAL          5G
(dlbids)       <nine, execfl, 02037>      LOCAL          5M
(dlckdl)       <nine, execfl, 02020>      LOCAL          5L
(dldriver)     <nine, execfl, 01094>      LOCAL          5A
(dlfmb)        <nine, execfl, 01236>      LOCAL          5C
(dlgetg)       <nine, execfl, 01737>      LOCAL          5H
(dlgets)       <nine, execfl, 01791>      LOCAL          5I
(dlgmb)        <nine, execfl, 01194>      LOCAL          5B
(dlgtblk)      <nine, execfl, 01260>      LOCAL          5D
(dlinfb)       <nine, execfl, 01858>      LOCAL          5J
(dlsnps)       <nine, execfl, 01989>      LOCAL          5K
(gerr)         <nine, execfl, 02086>      LOCAL          5N8
(getpcjfn)     <nine, execfl, 0389>       LOCAL          4E
(gtdesjfn)     <nine, execfl, 0994>       LOCAL          4P
(gtflgs)       <nine, execfl, 0203>       LOCAL          4A17A
(gtlftovr)     <nine, execfl, 0188>       LOCAL          4A16A
(gtver)        <nine, execfl, 0175>       LOCAL          4A15A
(isenabled)    <nine, execfl, 0455>       LOCAL          4F
(mkdirlist)    <nine, execfl, 01288>      LOCAL          5E
(movflandpc)   <nine, execfl, 0813>       LOCAL          4L
(parseinput)   <nine, execfl, 06>         LOCAL          4A
(parsil)       <nine, execfl, 02243>      LOCAL          4A14A
(proflandpc)   <nine, execfl, 0668>       LOCAL          4J
(proprot)      <nine, execfl, 0920>       LOCAL          4N
(sigstr)       <nine, execfl, 02209>      CATCHPHRASE    5Q10
(sigstr)       <nine, execfl, 02140>      CATCHPHRASE    5O9
(triflandpc)   <nine, execfl, 0581>       LOCAL          4I
(undflandpc)   <nine, execfl, 0530>       LOCAL          4H
(unprseinpt)   <nine, execfl, 0228>       LOCAL          4B
```

```
< NINE, EXECFL.NLS.8, >, 15-Feb-78 11:13 SKO ;;;;
FILE execfl    % <ARCSUBSYS>XL10  <RELNINE>execfl % %
(arcsubsys,xl10,) (arcsubsys,l109,) (RELNINE,execfl.rel,) %

ALLOW!
%.....Declarations.....%

%.....file manipulation commands' support routines.....%

   (parseinput) % parse input file link from user %                    4A
      PROCEDURE
         (fname,          % adr of user input file name %
         udirname,        % address of string to get directory name %
         ufilname,        % address of string to get file name %
         uextname,        % address of string to get extension name %
         uverno,          % address of string to get version number %
         ulftovr,         % address of string to get remaining input %
         uname, % address of string to get completed file name %
         flags  % bits saying which file fields had *s in them %
         );
      LOCAL
         bytptr % byte pointer for use with jsies %
         ;
      LOCAL TEXT POINTER
         tp1, tp2, tp3, dp1, dp2, fp1, fp2;
      LOCAL STRING
         locstr[200]    % local work string %
         ;
      REF
         fname, udirname, ufilname, uextname, uverno, ulftovr, uname,
         flags;

      % FUNCTION %
         % this procedure parses the input file link entered by a user.
         the following defaults are supplied:
            directory field defaults to null
               (this will force standard TENEX default of connected
               directory to apply.)
            file name field has no defaults and null is an error
            extension name field defaults to NLS if not supplied
            version number field defaults to *
               (this will cause all versions of a file to be dealt
               with)
            other fields default to null
               (this will force standard TENEX defaults to apply.)
         terminating a field with a ^F forces recognition of that
         specific field.
         terminating a field with a <ALT> forces recognition of that
         specific field and causes remaining unspecified fields to
         default to the above defaults.
         terminating a field with unspecified a space or an <EOL>
         completes that specific field and causes remaining fields to
         default to the above defaults. %

      % initialize locals %
         bytptr _ 0;
```

```
% do some initial parsing %
   FIND SF(*fname*) ^dp1 ^dp2 ^fp1 SE(*fname*) ^fp2;
   IF FIND dp1 >
      '< ^dp1 ['> ^dp2 _dp2 / ('<^F>/'<ESC>) ^dp2]  ^fp1 THEN
      NULL;
% set up local string to work with %
   *locstr* _ fp1 fp2;
% get directory name from directory input %
   *udirname* _ + dp1 dp2, 0;
   BUMP DOWN udirname.L;  % discount null byte at end %
   IF *udirname*[udirname.L] = $ascalt
      THEN *udirname*[udirname.L] _ $ctlf;
   % default to connected directory if not specified and not in
   NSW %
      IF NOT (nswsw OR udirname.L) THEN
         BEGIN    % get connected directory name string %
         !gjinf();
         gdname( R2, &udirname);
         *udirname* _ *udirname*, 0;
         BUMP DOWN udirname.L;    % discount null byte at end %
         END;
% get file name from input file name %
   FIND > SF(*locstr*) ^tp1 [ ('. / '; / '<^F> / '<ESC> / SP /
   EOL) ^tp2 _tp2 / ENDCHR ^tp2] ^tp3 ;
   CCPOS tp2;
   CASE READC OF
      = '.:
         BEGIN
         *ufilname* _ tp1 tp2, 0;
         BUMP DOWN ufilname.L;
         END;
      = ';:
         BEGIN
         *ufilname* _ tp1 tp2, 0;
         BUMP DOWN ufilname.L;
         *uextname* _ "NLS", 0;
         BUMP DOWN uextname.L;
         GOTO gtver;
         END;
      = '<^F>:
         BEGIN
         *ufilname* _ tp1 tp3, 0;
         BUMP DOWN ufilname.L;
         % file name of ^F only is equivalent to file name of * %
            IF ufilname.L = 1 THEN *ufilname*[1] _ '*;
         END;
      = '<ESC>:
         BEGIN
         *ufilname* _ tp1 tp2, '<^F>, 0;
         BUMP DOWN ufilname.L;
         % file name of only <ALT> is equivalent to *.NLS;* %
            IF ufilname.L = 1 THEN *ufilname*[1] _ '*;
         *uextname* _ "NLS", 0;
         BUMP DOWN uextname.L;
         *uverno* _ '*, 0;
         BUMP DOWN uverno.L;
```

```
            GOTO gtlftovr;
            END;
         = SP,
         = EOL:
            BEGIN
            *ufilname* _ tp1 tp2, 0;
            BUMP DOWN ufilname.L;
            *uextname* _ "NLS", 0;
            BUMP DOWN uextname.L;
            *uverno* _ '*, 0;
            BUMP DOWN uverno.L;
            GOTO gtflgs;
            END;
         = ENDCHR:
            BEGIN
            *ufilname* _ tp1 tp3, 0;
            BUMP DOWN ufilname.L;
            *uextname* _ "NLS", 0;
            BUMP DOWN uextname.L;
            *uverno* _ '*, 0;
            BUMP DOWN uverno.L;
            GOTO gtflgs;
            END;
         ENDCASE err( $"system screwup" );
   % get extension name from input file name %
      (parsl1):                                                4A14A
      FIND > tp3 ^tp1 [ ('; / '. / '<^F> / '<ESC> / SP / EOL) ^tp2
      _tp2 / ENDCHR ^tp2] ^tp3 ;
      CCPOS tp2;
      CASE READC OF
         = ';, = fvrdchar:
            BEGIN
            *uextname* _ tp1 tp2, 0;
            BUMP DOWN uextname.L;
            END;
         = '.: GOTO parsl1; %got second period but not running
         tops20%
         = '<^F>:
            BEGIN
            *uextname* _ tp1 tp3, 0;
            BUMP DOWN uextname.L;
            % ext name of ^F is equivalent to ext name of NLS %
               IF uextname.L = 1 THEN
                  BEGIN
                  *uextname* _ "NLS", 0;
                  BUMP DOWN uextname.L;
                  END;
            END;
         = '<ESC>:
            BEGIN
            *uextname* _ tp1 tp2, '<^F>, 0;
            BUMP DOWN uextname.L;
            % ext name of only <ALT> is equivalent to *.NLS;* %
               IF uextname.L = 1 THEN
                  BEGIN
                  *uextname* _ "NLS", 0;
                  BUMP DOWN uextname.L;
                  END;
              *uverno* _ '*, 0;
              BUMP DOWN uverno.L;
              GOTO gtlftovr;
              END;
          = SP,
          = EOL:
              BEGIN
              *uextname* _ tp1 tp2, 0;
              BUMP DOWN uextname.L;
              *uverno* _ '*, 0;
              BUMP DOWN uverno.L;
              GOTO gtflgs;
              END;
          = ENDCHR:
              BEGIN
              *uextname* _ tp1 tp3, 0;
              BUMP DOWN uextname.L;
              *uverno* _ '*, 0;
              BUMP DOWN uverno.L;
              GOTO gtflgs;
              END;
          ENDCASE err( $"system screwup" );
    % get version number string from input file name %
      (gtver):                                                          4A15A
      IF FIND > tp3 ^tp1 $1'- 1$D ^tp2 ^tp3 THEN
          BEGIN
          *uverno* _ tp1 tp2, 0;
          BUMP DOWN uverno.L;
          END
      ELSE
          BEGIN
          *uverno* _ '*, 0;
          BUMP DOWN uverno.L;
          IF FIND > tp1 ^tp3 ('<^F> / '<ESC> / '* ^tp3) THEN NULL;
          END;
    % now get any remaining user input %
      (gtlftovr):                                                       4A16A
      FIND > tp3 ^tp1 $1CH ^tp2 [ENDCHR] ^tp3 ;
      CCPOS tp1;
      CASE READC OF
          = ';, ='<^F>, ='<ESC>:
              BEGIN
              *ulftovr* _ tp2 tp3, 0;
              BUMP DOWN ulftovr.L;
              END;
          ENDCASE
              BEGIN
              *ulftovr* _ tp1 tp3, 0;
              BUMP DOWN ulftovr.L;
              END;
    % now set flags for which fields (if any) the user entered a * %
      (gtflgs):                                                         4A17A
      flags.dstar _
          IF udirname.L = 1 AND *udirname*[1] = '* THEN TRUE ELSE
```

```
         FALSE;
      flags.fstar _
         IF ufilname.L = 1 AND *ufilname*[1] = '* THEN TRUE ELSE
         FALSE;
      flags.estar _
         IF uextname.L = 1 AND *uextname*[1] = '* THEN TRUE ELSE
         FALSE;
      IF nswsw THEN
         BEGIN  %use of * not implemented in NSW%
         IF uverno.L = 1 AND *uverno*[1] = '* THEN
            *uverno*[1] _ '1;  %use version no. 1 as default for
            NSW%
         flags.vstar _ FALSE;
         END
      ELSE flags.vstar _
         IF uverno.L = 1 AND *uverno*[1] = '* THEN TRUE ELSE FALSE;
   % now build completed name %
      IF udirname.L THEN
         BEGIN  %insert directory name%
         *uname* _ '<, *udirname*;
         IF *uname*[uname.L] # '<^F> THEN *uname* _ *uname*, '>;
         END
      ELSE *uname* _ "";  %no directory name%
      *uname* _ *uname*, *ufilname*;
      IF *uname*[uname.L] # '<^F> THEN *uname* _ *uname*, '.;
      *uname* _ *uname*, *uextname*;
      IF *uname*[uname.L] # '<^F> THEN *uname* _ *uname*, fvrdchar;
      IF uverno.L THEN *uname* _ *uname*, *uverno*;
      IF ulftovr.L # 0 THEN *uname* _ *uname*, ';, *ulftovr*;
      *uname* _ *uname*, 0;
      BUMP DOWN uname.L;
   % all done now (i hope) %
      RETURN;

   END.

   %  %
```

```
(unprseinpt) % parse jfn into individual strings %                    4B
   PROCEDURE
      (fjfn,            % jfn of concerned file %
      jfnflgs,                  % left-half bits returned by gtjfn %
      udirname,         % address of string to receive directory name %
      ufilname,         % address of string to receive file name %
      uextname,         % address of string to receive extension name %
      uverno,                   % address of string to receive version
      number %
      ulftovr,                  % address of string to receive
      remaining stuff %
      uname,            % address of string to receive complete name %
      flags             % address of cell to receive * indicating bits
      %
      );
   LOCAL
      fuljfn            % jfn and flags %
      ;
   LOCAL TEXT POINTER
      tp1, tp2, tp3, tp4, tp5, tp6, tp7, tp8, tp9, tp10;
   REF udirname, ufilname, uextname, uverno, ulftovr, uname, flags;

   % initialize locals %
      fuljfn _ 0;
   % get complete name from jfns %
      fuljfn.LH _ jfnflgs;
      fuljfn.RH _ fjfn;
      jfntostr( fuljfn, &uname, 011111140001B);
   % set up pairs of text pointers to each field %
      IF NOT FIND SF(*uname*) > ['<] ^tp1 ['>] ^tp2 _tp2 ^tp3 ['.]
      ^tp4 _tp4 ^tp5 ['; / '.] ^tp6 _tp6 ^tp7 [';] ^tp8 _tp8 ^tp9
      [ENDCHR] ^tp10 THEN err( $"system screwup");
   % now build the individual strings %
      *udirname* _ + tp1 tp2, 0;  BUMP DOWN udirname.L;
      *ufilname* _ + tp3 tp4, 0;  BUMP DOWN ufilname.L;
      *uextname* _ + tp5 tp6, 0;  BUMP DOWN uextname.L;
      *uverno* _ + tp7 tp8, 0;  BUMP DOWN uverno.L;
      *ulftovr* _ + tp9 tp10, 0;  BUMP DOWN ulftovr.L;
   % now setup flags indicating which fields were astericks %
      flags.dstar _ jfnflgs.dstar;
      flags.fstar _ jfnflgs.fstar;
      flags.estar _ jfnflgs.estar;
      flags.vstar _ jfnflgs.vstar;
   % all done so return %
      RETURN;
   END.

   %  %
```

```
(chkdev)     % returns FALSE if not a supported (disk) file %      4C
   PROCEDURE
      (fjfn  % jfn of file to check %
      );
   LOCAL STRING
      devstring[40]  % string to get device string for this jfn %
      ;

   % get device name string from jfns %
      jfntostr(fjfn, $devstring, 100000B6);
   % now see if a disk file %
      RETURN( IF *devstring* = "DSK" THEN TRUE ELSE FALSE );

   END.

   %  %
```

```
(chkpcs)      % check if * for extname and extname = "PC" %           4D
   PROCEDURE
      (fjfn, % jfn of file to check %
      flags  % left half jfn flags of input %
      );
   LOCAL STRING
      pcname[40]      % temp string to get actual extension name %
      ;

   % RETURNS %
      % returns FALSE if * was input for the extension name and the
      extension name is "PC" (i.e. this file is a partial copy);
      returns true otherwise %

   % find out if * was input for extension name %
      IF NOT flags.estar THEN RETURN( TRUE );
   % now get the actual extension name %
      jfntostr( fjfn, $pcname, 000100B6 );
   % now check for extension name of "PC" %
      IF *pcname* = "PC" THEN RETURN( FALSE ) ELSE RETURN( TRUE );

   END.

   %  %
```

```
(getpcjfn)  % get jfn for pc of passed jfn %                        4E
   PROCEDURE
      (fjfn, % jfn of file %
      pcjfn, % address of variable to receive PC jfn %
      errstring,      % address of string to receive error messages %
      isok            % TRUE: ok to get pc if locked by someone else
      %
      );
   LOCAL
      temp,           % temp for pc name creation %
      dirnum,         % directory number of locking user %
      init,  % ident of locking user %
      ldirnum         % login directory number for this user %
      ;
   LOCAL STRING
      dirstring[40], % temp string to check for valid locking dir. #
      %
      fstring[200],  % string to get file name for passed file jfn %
      pcstring[200]  % string to get partial copy name %
      ;
   REF pcjfn, errstring;

   % FUNCTION %
      % get a jfn for the partial copy (if one exists) for the
      passed jfn %
   % RETURNS %
      % returns TRUE if no partial copy exists, or if partial copy
      exists and the file is locked by this user, or if partial copy
      exists and isok parameter is TRUE, or if a partial copy exists
      and this user is an enabled wheel, or if the file is the
      TOPS20 MAIL.TXT file (which uses the user-settable word);
      returns FALSE otherwise. %

   % initialize locals %
      dirnum _ init _ ldirnum _ 0;
   % initialize error string %
      *errstring* _ NULL;
   % initially there is no pc jfn %
      pcjfn _ 0;
   % now get user settable word to determine if pc exists %
      !gtfdb( fjfn, 1B6+24B, $R3);
   % return true if no pc %
      IF (dirnum _ R3.lkdirn) = 0 THEN RETURN( TRUE );
   % get ident of locking user %
      init _ R3.lkinit;
   %check if MAIL.TXT file on TOPS20%
      IF tops20flag THEN
         BEGIN
         jfnstr(fjfn, $fstring);
         IF FIND SF(*fstring*) [">MAIL.TXT."] THEN RETURN(TRUE);
         END;
   % get login directory of this user %
      !gjinf();
      ldirnum _ R1;
   % see if locked by this user %
      IF NOT ( (dirnum = ldirnum) AND (init = cinit) ) THEN
```

```
        BEGIN      % locked by someone else %
        % if invalid locking dir. # then not an NLS file, return
        TRUE %
           IF NOT SKIP !dirst( chbmty+$dirstring, dirnum) THEN
           RETURN( TRUE );
        % ok to let user have this file even tho locked ? %
           IF NOT isok THEN
              % find out if this is an enabled wheel %
                 IF NOT isenabled() THEN
                    BEGIN   % not an enabled wheel %
                    % create error message %
                       lockid ( &errstring, dirnum, init);
                    % now return FALSE %
                       RETURN( FALSE );
                    END;
        END;
  % now get string name for partial copy %
     jfnstr( fjfn, $fstring);
     temp _ cpcnam( $fstring, dirnum);
  % now get a jfn for the partial copy %
     pcjfn _ sgtjfn( gtjprv+gtjoif+gtjidl, temp, &errstring);
  % free the string obtained in getstring and return %
     freestring( temp, $dspblk);
     RETURN( pcjfn );
  END.

  %  %
```

```
(isenabled) % determine if this user is an enabled wheel %           4F
   PROCEDURE;

   % RETURNS %
      % returns TRUE if this user is an enabled wheel; FALSE
      otherwise %

   % get enabled capabilites for this process %
      !rpcap( 400000B );
   % now return true or false depending on state of this user %
      IF R3 .A 4B5 THEN RETURN( TRUE ) ELSE RETURN( FALSE );

   END.

   % %
```

```
(delflandpc) % delete file (and partial copy) for input jfns %      4G
   PROCEDURE
      (fjfn, % jfn of file %
      pcjfn, % jfn of PC (or 0) %
      erstrng         % address of string to receive error messages %
      );
   LOCAL
      arstus,         % TRUE means archive pending for the file %
      delmode         % FALSE means pc was already deleted %
      ;
   REF erstrng;

   % ALGORITHM %
      % deletes files by changing the deleted bit in the FDB %
   % RETURNS %
      % returns TRUE if file(s) deleted properly.  On FALSE returns,
      writes erstrng with reason for failure and nothing is deleted
      %

   % initialize locals %
      delmode _ 0;
   % delete partial copy first if it exists %
      IF pcjfn THEN
         BEGIN        % find out if already deleted %
         !gtfdb( pcjfn, 1B6 + 1B, $R3);
         IF NOT R3 .A 4B10 THEN
            BEGIN    % not deleted yet %
            delmode _ TRUE;
            !gtfdb( pcjfn, 1B6+$fdbbkf, $R3);
            IF R3.flarfl.fdbarc THEN
               BEGIN % can't delete archive pending partial copy %
               *erstrng* _ "can't delete archive pending partial
               copy";
               RETURN( FALSE );
               END;
            IF NOT chnfdb( pcjfn, 1B, 4B10, 4B10) THEN
               BEGIN % can't delete partial copy %
               *erstrng* _ "can't delete partial copy";
               RETURN( FALSE );
               END;
            END
         ELSE delmode _ FALSE;        % already deleted %
         END;
   % now delete the file itself %
      !gtfdb( fjfn, 1B6+$fdbbkf, $R3);
      arstus _ R3.flarfl.fdbarc;
      IF arstus OR NOT chnfdb( fjfn, 1B, 4B10, 4B10) THEN
         BEGIN        % can't delete the file %
         IF arstus THEN
            *erstrng* _ "can't delete archive pending file"
         ELSE *erstrng* _ "can't delete this file";
         % now restore partial copy to previous state %
            IF delmode THEN
               % we deleted the partial copy above %
                  IF NOT chnfdb( pcjfn, 1B, 4B10, 0) THEN
                     % we could delete above, but cant undelete here
```

```
                        %
                        err( $"system screwup" );
            % now give false return since we cant delete the file %
              RETURN( FALSE );
            END;
        % we're all done now so return true %
          RETURN( TRUE );

        END.

        %  %
```

```
(undflandpc) % undelete file (and partial copy) for input jfns %   4H
  PROCEDURE
     (fjfn, % jfn of file %
     pcjfn, % jfn of PC (or 0) %
     erstrng        % address of string to receive error messages %
     );
   LOCAL
     delmode        % FALSE means pc was already undeleted %
     ;
   REF errstring;

   % ALGORITHM %
     % undeletes files by changing the deleted bit in the FDB %
   % RETURNS %
     % returns TRUE if file(s) undeleted properly.  On FALSE
     returns, writes erstrng with reason for failure and nothing is
     undeleted %

   % initialize locals %
     delmode _ 0;
   % undelete partial copy first if it exists %
     IF pcjfn THEN
        BEGIN       % find out if already undeleted %
        !gtfdb( pcjfn, 1B6 + 1B, $R3);
        IF R3 .A 4B10 THEN
           BEGIN    % not undeleted yet %
           delmode _ TRUE;
           IF NOT chnfdb( pcjfn, 1B, 4B10, 0) THEN
              BEGIN % can't undelete partial copy %
              *erstrng* _ "can't undelete partial copy";
              RETURN( FALSE );
              END;
           END
        ELSE delmode _ FALSE;       % already undeleted %
        END;
   % now undelete the file itself %
     IF NOT chnfdb( fjfn, 1B, 4B10, 0) THEN
        BEGIN       % can't undelete the file %
        *erstrng* _ "can't undelete this file";
        % now restore partial copy to previous state %
           IF delmode THEN
              % we undeleted the partial copy above %
                 IF NOT chnfdb( pcjfn, 1B, 4B10, 4B10) THEN
                    % we could undelete above, but cant delete here
                    %
                    err( $"system screwup" );
        % now give false return since we cant undelete the file %
           RETURN( FALSE );
        END;
   % we're all done now so return true %
     RETURN( TRUE );

   END.

   %  %
```

```
(triflandpc) % trim file (and partial copy) for input jfns %        4I
   PROCEDURE
      (fjfn, % jfn of file %
      pcjfn, % jfn of PC (or 0) %
      nver,  % number of versions to keep for each file %
      erstrng        % address of string to receive error messages %
      );
   LOCAL
      icount,        % loop index for untrimming %
      pcgjfn,        % group jfn for untrimming partial copies %
      jfnflgs,       % left half gtjfn flags %
      delmode        % FALSE means pc was already deleted %
      ;
   LOCAL STRING
      jfnname[200],  % string for partial copy untrimming %
      xstring[100]   % error string for sgtjfn %
      ;
   REF erstrng;

   % ALGORITHM %
      % TRIMS files by using the DELNF jsys %
   % RETURNS %
      % returns TRUE (and number of files deleted) if file(s)
      trimmed properly.  On FALSE returns, writes erstrng with
      reason for failure and nothing is trimmed %

   % initialize locals %
      delmode _ 0;
   % initialize error string %
      *erstrng* _ NULL;
   % trim partial copy first if it exists %
      IF pcjfn THEN
         IF NOT SKIP !delnf( pcjfn, nver) THEN
            BEGIN
            *erstrng* _ "can't trim partial copy";
            RETURN( FALSE, 0);
            END
         ELSE delmode _ (IF tops20flag THEN R2 ELSE -R2); %number of
         versions deleted %
   % now trim the file itself and return %
      IF NOT SKIP !delnf( fjfn, nver) THEN
         BEGIN         % can't trim the file %
         *erstrng* _ "can't trim this file";
         % now restore partial copy to previous state %
            IF delmode > 0 THEN
               BEGIN % we trimmed the partial copy above %
               % get string name for undelete loop %
                  jfntostr( pcjfn, $jfnname, 011100B6+1);
                  *jfnname* _ *jfnname*, fvrdchar, '*, 0;
                  BUMP DOWN jfnname.L;
               % now get a group jfn for it %
                  IF NOT (pcgjfn _ sgtjfn( gtjoif .V gtjstr,
                  $jfnname, $xstring : jfnflgs) ) THEN
                     BEGIN
                     *erstrng* _ *erstrng*, CR, LF, "   ***   can't
                     untrim trimmed partial copies ***";
```

```
                    RETURN( FALSE, 0 );
                    END;
              % now loop past untrimmed files first %
                 FOR lcount _ 1 UP UNTIL >= nver DO
                    BEGIN
                    R1.LH _ jfnflgs;
                    R1.RH _ pcgjfn;
                    IF NOT SKIP !gnjfn( R1 ) THEN
                       BEGIN
                       IF NOT SKIP !rljfn( pcgjfn ) THEN NULL;
                       *erstrng* _ *erstrng*, CR, LF, "   ***
                       can't untrim trimmed partial copies ***";
                       RETURN( FALSE, 0);
                       END;
                    END;
              % now untrim partial copies trimmed above %
                 FOR lcount _ 1 UP UNTIL > delmode DO
                    BEGIN
                    R1.LH _ jfnflgs;
                    R1.RH _ pcgjfn;
                    IF NOT SKIP !gnjfn( R1 ) THEN
                       BEGIN
                       IF NOT SKIP !rljfn( pcgjfn ) THEN NULL;
                       RETURN( FALSE, 0 );
                       END;
                    % now untrim one version %
                       IF NOT chnfdb( pcgjfn, 1B, 4B10, 0) THEN
                          *erstrng* _ *erstrng*, CR, LF, "   ***
                          can't untrim trimmed partial copies ***";
                    END;
              END;
        % now give false return since we cant undelete the file %
           RETURN( FALSE, 0 );
        END
    ELSE RETURN( TRUE, (IF tops20flag THEN R2 ELSE -R2));

  END.

  %  %
```

```
(profandpc) % set protection of a file (and its pc) %              4J
   PROCEDURE
      (fjfn, % jfn of the file %
      pcjfn, % jfn of the pc or zero %
      mask,  % mask indicating which bits to change %
      prot,  % new protection bits %
      astr   % address of string to get error message %
      );
   LOCAL
      newprot,        %computed new protection%
      pcprot % old protection of the pc %
      ;
   REF astr;

   % initial variables %
      pcprot _ 0;
   % mask only the right half of the protection word %
      mask.LH _ 0;
   % change protection of the partial copy first %
      IF pcjfn THEN
         BEGIN
         % get the old protection %
            !gtfdb( pcjfn, 1000004B, $pcprot);
         % change the protection %
            IF pcprot.LH # 500000B THEN
               BEGIN % if fancy protection reset first %
               newprot  _ (500000777752B .A (mask .X -1)) .V (prot
               .A mask);
               IF NOT chnfdb( pcjfn, 4, 36M, newprot) THEN
                  BEGIN
                  *astr* _ "can't change protection of partial
                  copy";
                  RETURN( FALSE );
                  END;
               END
            ELSE
               IF NOT chnfdb( pcjfn, 4, mask, prot) THEN
                  BEGIN
                  *astr* _ "can't change protection of partial
                  copy";
                  RETURN( FALSE );
                  END;
         END;
   % now do the file %
      % get the old protection %
         !gtfdb( fjfn, 1000004B, $newprot);
      % change the protection %
         IF newprot.LH # 500000B THEN
            BEGIN    % if fancy protection reset first %
            newprot  _ (500000777752B .A (mask .X -1)) .V (prot .A
            mask);
            IF NOT chnfdb( fjfn, 4, 36M, newprot) THEN
               BEGIN
               *astr* _ "can't change protection of the file";
               % reset the pc protection if changed above %
                  IF pcprot THEN
```

```
                    BEGIN
                    IF NOT chnfdb( pcjfn, 4, 36M, pcprot) THEN
                       *astr* _ "file protection not changed;
                       partial copy protection changed";
                    END;
                RETURN( FALSE );
                END;
             END
          ELSE
             IF NOT chnfdb( fjfn, 4, mask, prot) THEN
                BEGIN
                *astr* _ "can't change protection of the file";
                % reset the pc protection if changed above %
                  IF pcprot THEN
                    BEGIN
                    IF NOT chnfdb( pcjfn, 4, 36M, pcprot) THEN
                       *astr* _ "file protection not changed;
                       partial copy protection changed";
                    END;
                RETURN( FALSE );
                END;
  % all done, so return %
     RETURN( TRUE );

  END.

  %  %
```

```
(arcflandpc) % set archive status of a file (and its pc) %          4K
   PROCEDURE
      (fjfn, % jfn of the file %
      pcjfn, % jfn of the pc or zero %
      arcparms,       % new archive status bits %
      astr   % address of string to get error message %
      );
   LOCAL
      mask,  % mask indicating which bits to change %
      arstus,        % old archive status of the file %
      pcarc  % old archive status of the pc %
      ;
   REF astr;

   % initial variables %
      *astr* _ NULL;
      pcarc _ 0;
   % set up mask for which bits we will change %
      mask _ FALSE;
      mask.flarfl.fdbadl _ TRUE;
   % change archive status of the partial copy first %
      IF pcjfn THEN
         BEGIN
         % get the old archive status %
            !gtfdb( pcjfn, 1B6+$fdbbkf, $pcarc);
         % change the archive status %
            IF pcarc.flarfl.fdbaar THEN
               BEGIN
               mask.flarfl.fdbnar _ FALSE;
               mask.flarfl.fdbarc _ FALSE;
               *astr* _ "partial copy already archived";
               END
            ELSE
               BEGIN
               mask.flarfl.fdbnar _ TRUE;
               mask.flarfl.fdbarc _ TRUE;
               END;
            IF NOT chnfdb( pcjfn, $fdbbkf, mask, arcparms) THEN
               BEGIN
               *astr* _ "[can't change archive status of partial
               copy]";
               RETURN( FALSE );
               END;
         END;
   % now do the file %
      % get the old archive status %
         !gtfdb( fjfn, 1B6+$fdbbkf, $R3);
         arstus _ R3.flarfl.fdbaar;
      % change the archive status %
         IF arstus THEN
            BEGIN
            mask.flarfl.fdbnar _ FALSE;
            mask.flarfl.fdbarc _ FALSE;
            IF astr.L THEN *astr* _ "file and ", *astr*
            ELSE *astr* _ "file already archived";
            END
         ELSE
            BEGIN
            mask.flarfl.fdbnar _ TRUE;
            mask.flarfl.fdbarc _ TRUE;
            IF astr.L THEN *astr* _ *astr*, "; archive status
            changed on original file only";
            END;
         IF NOT chnfdb( fjfn, $fdbbkf, mask, arcparms) THEN
            BEGIN
            *astr* _ "can't change archive status of the file";
            % reset the pc archive status if changed above %
               IF pcjfn THEN
                  BEGIN
                  IF NOT chnfdb( pcjfn, fdbbkf, -1, pcarc) THEN
                     *astr* _ "file archive status not changed;
                     partial copy archive status changed";
                  END;
            RETURN( FALSE );
            END;
   % all done, so return %
      RETURN( TRUE );

   END.

   %  %
```

```
(movfiandpc) % do the move file stuff %                                    4L
   PROCEDURE
      (jfn1, % jfn for source file %
      pcjfn1,         % jfn for source file pc %
      jfn2,  % jfn for destination file %
      pcjfn2,         % jfn for destination file pc %
      errstring);     % address of string to get error messages %
   LOCAL movmode, fjfn, fpcjfn;
   LOCAL STRING filnam[200], pcnam[200];
   REF errstring;

   % initialize %
      movmode _ TRUE;
   % get local jfns since rnamf will release our group jfn %
      jfnstr( jfn1, $filnam);
      IF NOT fjfn _ sgtjfn( gtjoif, $filnam, &errstring) THEN
         BEGIN
         movmode _ FALSE;
         *errstring* _ "Can't GTJFN for the file";
         END;
      IF pcjfn1 THEN
         BEGIN
         jfnstr( pcjfn1, $pcnam);
         IF NOT fpcjfn _ sgtjfn( gtjoif, $pcnam, &errstring) THEN
            BEGIN
            movmode _ FALSE;
            *errstring* _ "[Can't GTJFN for the PC]";
            END;
         END
      ELSE fpcjfn _ FALSE;
   IF movmode THEN
      BEGIN
      % move partial copy first if it exists %
         IF fpcjfn THEN
            BEGIN
            IF NOT SKIP !rnamf( fpcjfn, pcjfn2 ) THEN
               BEGIN
               *errstring* _ "[can't move partial copy]";
               movmode _ FALSE;
               END;
            % propagate protection to new file %
               proprot( pcjfn1, pcjfn2);
            END;
      % now move the file itself %
         IF movmode THEN
            IF NOT SKIP !rnamf( fjfn, jfn2 ) THEN
               BEGIN
               movmode _ FALSE;
               *errstring* _ "can't move this file";
               % now unmove partial copy if needed %
                  IF fpcjfn THEN
                     IF NOT SKIP !rnamf( pcjfn2, pcjfn1 ) THEN
                        err( $"system screwup" );
               END;
         % propagate protection to new file %
            proprot( jfn1, jfn2);
      END;
   % get rid of extraneous jfns %
      IF NOT SKIP !rljfn(jfn2) THEN NULL;
      IF NOT SKIP !rljfn(fjfn) THEN NULL;
      IF pcjfn2 THEN
         BEGIN
         IF NOT SKIP !rljfn(fpcjfn) THEN NULL;
         IF NOT SKIP !rljfn(pcjfn2) THEN NULL;
         END;
   % we're all done now so return %
      RETURN( movmode );
   END.

   %  %
```

```
(copflandpc) % do the Copy file stuff %                                    4N
   PROCEDURE
      (jfn1, % jfn for source file %
      pcjfn1,         % jfn for source file pc %
      jfn2,  % jfn for destination file %
      pcjfn2,         % jfn for destination file pc %
      errstring);     % address of string to get error messages %
   LOCAL copmode;
   LOCAL STRING filnam[200], pcnam[200];
   REF errstring;

   % initialize %
      copmode _ TRUE;
   % copy partial copy first if it exists %
      IF pcjfn1 THEN
         IF NOT cflapc( pcjfn1, pcjfn2 ) THEN
            BEGIN
            *errstring* _ "[can't copy partial copy]";
            copmode _ FALSE;
            END;
   % now copy the file itself %
      IF copmode THEN
         IF NOT cflapc( jfn1, jfn2 ) THEN
            BEGIN
            copmode _ FALSE;
            *errstring* _ "can't copy this file";
            % now delete copied partial copy if needed %
               IF pcjfn1 THEN
                  IF NOT SKIP !delf( pcjfn2 ) THEN
                     err( $"system screwup" );
            END;
   % get rid of extraneous jfns %
      IF NOT SKIP !rljfn(jfn2) THEN NULL;
      IF pcjfn2 THEN IF NOT SKIP !rljfn(pcjfn2) THEN NULL;
   % we're all done now so return %
      RETURN( copmode );
   END.

   %  %
```

```
(proprot)    % propagate protection from jfn1 to jfn2 %                4N
   PROCEDURE( jfn1, jfn2);
   % get protection of old file %
      !gtfdb( jfn1, 1B6 + $fdbprt, $R3);
   % now set protection for new file %
      IF R3.LH = 500000B THEN chnfdb( jfn2, $fdbprt, 18M, R3);
   RETURN;
   END.

   %  %
```

```
(cflapc)      % copy file from jfn1 to jfn2 %                        40
   PROCEDURE( jfn1, jfn2);
   LOCAL page, bytsiz, filsiz, mask;

   % initialize %
      page _ -1;
   % open the source file %
      IF NOT sysopen( jfn1, read, bintyp, $lit) THEN RETURN( FALSE
      );
   % get byte size and length of source file %
      !gtfdb( jfn1, 1B6 + $fdbbyv, $bytsiz);
         mask _ 0;
         bytsiz _ mask.flbyts _ bytsiz.flbyts;
      !gtfdb( jfn1, 1B6 + $fdbsiz, $filsiz);
   % open the destination file %
      IF NOT SKIP !openf( jfn2, (bytsiz * 1B10) .V 1B5) THEN
         BEGIN
         IF NOT SKIP !closf(4B11 + jfn1) THEN NULL;
         RETURN( FALSE );
         END;
   % set the size of the destination file %
      IF NOT chnfdb( jfn2, $fdbsiz, 36M, filsiz) THEN NULL;
      bytsiz _ 0;  bytsiz.flbyts _ mask.flbyts;
      mask.flbyts _ -1;
      IF NOT chnfdb( jfn2, $fdbbyv, mask, bytsiz) THEN NULL;
   % loop to get all pages copied %
      LOOP
         BEGIN
         % find used file page %
            R1.LH _ jfn1;
            R1.RH _ page + 1;
            IF NOT SKIP !ffufp( R1 ) THEN EXIT LOOP;
            page _ R1.RH;
         % map in this page %
            % R1.LH _ jfn1;  R1.RH _ page;  R1 still setup from
            ffufp %
            R2.LH _ 4B5;  R2.RH _ $ckpag/1000B;
            R3 _ 1B11;          % read access %
            !pmap( R1, R2, R3);
         % copy the page %
            R1.LH _ $ckpag;  R1.RH _ $sfbuff;
            R2 _ $sfbuff+511;
            !BLT R1,(R2);
         % unmap the source page %
            R1 _ -1;
            R2.LH _ 4B5;  R2.RH _ $ckpag/1000B;
            R3 _ 0;
            !pmap( R1, R2, R3);
         % map destination page to destination file %
            R1.LH _ 4B5;  R1.RH _ $sfbuff/1000B;
            R2.LH _ jfn2;  R2.RH _ page;
            R3 _ 1B11;
            !pmap( R1, R2, R3);
         % unmap the destination page %
            R1 _ -1;
            R2.LH _ 4B5;  P2.RH _ $sfbuff/1000B;
            R3 _ 0;
            !pmap( R1, R2, R3);
         END;
   % propagate protection to new file %
     proprot( jfn1, jfn2);
   % close the files (don't release the jfns) %
     IF NOT !closf( 4B11 + jfn1 ) THEN NULL;
     IF NOT !closf( 4B11 + jfn2 ) THEN NULL;
   RETURN( TRUE );
   END.

   %  %
```

```
(gtdesjfn)   % get destination jfns  ! Can't get JFN if there is
already a deleted file ! %                                                4P
   PROCEDURE
      (jfn1, % source file jfn %
      pcjfn1,         % source file pc jfn %
      udir2, % destination directory name %
      ufil2, % destination file name %
      uext2, % destination extension name %
      uver2, % destination version number %
      ulft2, % destination miscellaneous stuff %
      flag2, % destination star field flag bits %
      errstring      % error string address %
      );
   LOCAL oldver, dver, jfn2, pcjfn2, dirnum, temp, flag1, mask,
   value;
   LOCAL TEXT POINTER dp1, dp2, fp1, fp2;
   LOCAL STRING
      filnam[200], fil2lnk[200], trash[200], uf1[200], udir1[40],
      ufil1[40], uext1[40], uver1[40], ulft1[40];
   REF udir2, ufil2, uext2, uver2, ulft2, errstring;

   % initialize %
      pcjfn2 _ jfn2 _ 0;
   % create destination filename for gtjfn %
      IF NOT flag2 THEN
         BEGIN
         *filnam* _ '<, *udir2*, '>, *ufil2*, '., *uext2*, fvrdchar,
         *uver2*;
         IF ulft2.L THEN *filnam* _ *filnam*, ';, *ulft2*;
         END
      ELSE
         BEGIN
         unprseinpt( jfn1, 0, $udir1, $ufil1, $uext1, $uver1,
         $ulft1, $uf1, $flag1);
         *filnam* _ '<;
         CASE flag2.dstar OF
            = FALSE: *filnam* _ *filnam*, *udir2*, '>;
            ENDCASE *filnam* _ *filnam*, *udir1*, '>;
         CASE flag2.fstar OF
            = FALSE: *filnam* _ *filnam*, *ufil2*, '.;
            ENDCASE *filnam* _ *filnam*, *ufil1*, '.;
         CASE flag2.estar OF
            = FALSE: *filnam* _ *filnam*, *uext2*, fvrdchar;
            ENDCASE *filnam* _ *filnam*, *uext1*, fvrdchar;
         CASE flag2.vstar OF
            = FALSE: *filnam* _ *filnam*, *uver2*;
            ENDCASE *filnam* _ *filnam*, *uver1*;
         CASE ulft2.L OF
            = 0: NULL;
            = 1:
               CASE *ulft2*[1] OF
                  = '*: *filnam* _ *filnam*, ';, *ulft1*;
                  ENDCASE *filnam* _ *filnam*, ';, *ulft2*;
            ENDCASE *filnam* _ *filnam*, ';, *ulft2*;
         END;
   % create link name for error reporting %
```

```
        FIND SF(*filnam*) ['<] ^dp1 ['>] ^fp1 ^dp2 _ dp2 SE(*filnam*)
        ^fp2;
        *fil2lnk* _ "< ", + dp1 dp2, ", ", + fp1 fp2, ", >";
     % gtjfn for destination file %
        jfn2 _ sgtjfn(gtjidl, $filnam, &errstring);
        IF jfn2 THEN
           BEGIN
           !gtfdb(jfn2,1B6+$fdbctl, $R3);
           IF NOT (R3.fdbnxf OR R3.fdbdel) THEN
              BEGIN
              IF NOT SKIP !rljfn(jfn2) THEN NULL;
              *errstring* _ *fil2lnk*, " already exists.";
              RETURN(FALSE, FALSE);
              END;
           END
        ELSE
           BEGIN
           *errstring* _ "Can't GTJFN for ", *fil2lnk*;
           RETURN(FALSE, FALSE);
           END;
     % gtjfn for PC for destination if source has a PC %
        IF NOT pcjfn1 THEN RETURN( jfn2, 0);
        % get login directory number %
           !gjinf();
           dirnum _ R1;
        % try to lock the destination file %
           mask _ value _ 0;
           mask.lkinit _ mask.lkdirn _ 36M;
           value.lkinit _ cinit;
           value.lkdirn _ dirnum;
           IF NOT chnfdb( jfn2, 24B, mask, value) THEN
              BEGIN
              IF NOT !rljfn(jfn2 := 0) THEN NULL;
              *errstring* _ "[Can't Lock ", *fil2lnk*, '];
              RETURN( FALSE, FALSE);
              END;
        % get address of string containing name for new PC %
           temp _ cpcnam( $filnam, dirnum);
        % get jfn for new PC %
           pcjfn2 _ sgtjfn( gtjnfo+gtjidl, temp, &errstring);
        % free the string gotten in cpcnam %
           freestring( temp, $dspblk);
        % report any errors %
           IF NOT pcjfn2 THEN
              BEGIN
              IF NOT !rljfn(jfn2 := 0) THEN NULL;
              *errstring* _ "[Can't GTJFN for partial copy for ",
              *fil2lnk*, '];
              RETURN( FALSE, FALSE);
              END;
     % all done, so return %
        RETURN( jfn2, pcjfn2);
     END.

     %  %
```

```
%.....copy and show directory utility subroutines....%
   (dldriver)   % main procedure for directory commands %                5A
      PROCEDURE
         (info, % record desribing what information requested %
         gropk, % record desribing how to group things %
         sortk, % record describing how to sort things %
         fjfn,            % driving jfn %
         jfnflgs,         % leht half flags for gnjfn %
         erstrng,         % string for error messages %
         adlmb  % address of first directory list master block %
         );
      LOCAL
         tptr,            % temp pointer %
         pcjfn, % jfn of the pc if it exists %
         delbit,          % deleted, undeleted, or all type request %
         nlschn,          % chain pointer to files w/ pc for 2nd pass %
         pcchn, % chain pointer to pc files for 2nd pass %
         chns,  % contains address of data structure chain start %
         adlfb  % address of directory list file blocks %
         ;
      REF tptr, erstrng, nlschn, pcchn, adlfb, adlmb;

      % initialize locals %
         &pcchn _ &nlschn _ chns _ &adlfb _ 0;
         delbit _ info.dlidlt := FALSE;
      % turn off the FOR FILE field in the info record %
         info.dlifrf _ FALSE;
      % loop to get all files %
         LOOP
            BEGIN
            % exit if control-O interrupt %
               IF inptrf THEN EXIT LOOP;
            % see if this file has proper deletion status %
               IF NOT dlckdl( fjfn, delbit) THEN
                  IF NOT SKIP !gnjfn(jfnflgs * 1B6 + fjfn) THEN
                     BEGIN
                     % dont return null chain in this case %
                        IF NOT chns THEN chns _ dlgtblk( &adlmb,
                        dlgbl);
                     EXIT LOOP;
                     END
                  ELSE REPEAT LOOP;
            % get pc jfn if it exists %
               erstrng.L _ 0;
               getpcjfn( fjfn, $pcjfn, $erstrng, TRUE);
            % get directory list file block for this file (and pc) %
               IF NOT
                  (&adlfb _ mkdirlist(fjfn, pcjfn, info, gropk, sortk,
                  &adlmb))
                  THEN
                     BEGIN
                     *erstrng* _ "Ran Out Of Space Before Finishing";
                     % get rid of redundant pc files %
                        dlsnps( &nlschn, &pcchn);
                     RETURN( chns );
                     END;
```

```
        % add this dlfb to the data structure %
           IF NOT (dlinfb($chns, &adlmb, &adlfb, sortk)) THEN
              BEGIN
              *erstrng* _ "Ran Out Of Space Before Finishing";
              % get rid of redundant pc files %
                 dlsnps( &nlschn, &pcchn);
              RETURN( chns );
              END;
        % handle any errors from getpcjfn above %
           IF erstrng.L THEN
              IF ( &tptr _
                 dlgtblk( &adlmb, [adlfb.dlfbal].L+erstrng.L+1+2) )
                 THEN
                    BEGIN
                    tptr.M _ [adlfb.dlfbal].L + erstrng.L + 4;
                    *tptr* _ *[adlfb.dlfbal]*, "  [", *erstrng*,
                    ']';
                    adlfb.dlfbal _ &tptr;
                    END
              ELSE
                 BEGIN
                 *erstrng* _ "Ran Out Of Space Before Finishing";
                 % get rid of redundant pc files %
                    dlsnps( &nlschn, &pcchn);
                 RETURN( chns );
                 END;
        % create second pass chains %
           IF adlfb.dlfbfl.dlstpc THEN
              BEGIN
              adlfb.dlfbcp _ &pcchn;
              &pcchn _ &adlfb;
              END
           ELSE IF adlfb.dlfbfl.dlstnl THEN
              BEGIN
              adlfb.dlfbcp _ &nlschn;
              &nlschn _ &adlfb;
              END;
        % go on to the next file %
           IF pcjfn THEN IF NOT SKIP !rljfn( pcjfn ) THEN NULL;
           IF NOT SKIP !gnjfn( jfnflgs * 1B6 + fjfn ) THEN EXIT
           LOOP;
        END;
  % get rid of redundant pc files %
     dlsnps( &nlschn, &pcchn);
  % i think we're done so i'll return %
     RETURN( chns );

  END.

  %  %
```

```
(dlgmb)      % get and initialize a directory list master block %  5B
   PROCEDURE
      (blksiz);       % size of desired block %                      5B1A
   LOCAL
      temp,           % temp for use in allocating file pages %
      block; % address of allocated block %
   REF block;

   % FUNCTION %
      % this procedure gets and set up a directory list master block
      of core that can be used to allcate the needed blocks for the
      directory (and similar) commands. %
   % RETURNS %
      % returns the address of the allocated master block on
      succesful completion or FALSE on unsuccesfull completion %

   % get a block (or file pages) and initialize the storage %
      IF NOT (&block _ getblk( blksiz, $dspblk) - bhl ) THEN
         BEGIN        % ran out of block storage: use file pages %
         IF NOT oldpgsz THEN temp _ upgbsz;
         IF NOT &block _
            gpbsz( MAX(upgbsz+2, upgbsz+((blksiz+511)/512)+1 ) )
            THEN RETURN( FALSE );
         IF NOT oldpgsz THEN oldpgsz _ temp;
         % zero the new pages %
            block _ 0;
            R1.LH _ &block;
            R1.RH _ &block + 1;
            !BLT R1,upgbend;
         % now set it up to look like regular block %
            block.dlmbln _ block.dlmbal _ upgbend-&block-dlmbl-1;
            block.dlmbfp _ &block + dlmbl;
         END
      ELSE
         BEGIN
         % now initialize the block %
            block.dlmbzn _ $dspblk;
            block.dlmbnm _ 0;
            block.dlmbfp _ &block + dlmbl;
         END;
   % now return %
      RETURN( &block );

   END.

   %  %
```

```
(dlfmb)       % free directory list master blocks %                    5C
   PROCEDURE
      (ablk  % address of first master block %
      );
   LOCAL
      temp;
   REF ablk;

   % ALGORITHM %
      % frees all master blocks in one chain starting with the
      master block whose address is passed %

   % free all blocks in the chain and return %
      WHILE &ablk DO
         BEGIN
         temp _ ablk.dlmbnm;
         IF ablk.dlmbzn THEN freeblk( &ablk + bhl, ablk.dlmbzn );
         &ablk _ temp;
         END;
      % now free any file pages %
         IF oldpgsz THEN gpbsz(oldpgsz := 0);
      RETURN;

   END.

   %  %
```

```
(digtblk)     % get a directory block of length blksiz %                 5D
   PROCEDURE
      (ablk,  % master block address within which to allocate block
      %
      blksiz  % desired block size %
      );
   LOCAL
      temp;
   REF ablk;

   % RETURNS %
      % returns the address of the allocated storage block or FALSE
      %

   % if room for new block, update free pointer and return %
      LOOP
         IF ((ablk.dlmbfp - &ablk + blksiz) < ablk.dlmbln)
            THEN RETURN ( ablk.dlmbfp := ablk.dlmbfp + blksiz )
         ELSE
            IF ablk.dlmbnm # 0 THEN &ablk _ ablk.dlmbnm
            ELSE
               BEGIN
               IF NOT ( temp_dlgmb(MAX(ablk.dlmbln, blksiz+dlmbl+1))
               )
                  THEN IF NOT ( temp_dlgmb(blksiz+dlmbl+1) ) THEN
                     RETURN( FALSE );
               &ablk _ ablk.dlmbnm _ temp;
               END;

   END.

   %  %
```

```
(mkdirlist)  % make an entry for a directory list %                    5E
   PROCEDURE
      (fjfn, % jfn of file to be listed %
      pcjfn, % jfn of partial copy for file if it exists %
      info,  % record (dlinfo) for which fields are to be listed %
      gropk, % record (dlgrp) for what grouping is to be done %
      sortk, % record (dlsort) for what sorting is to be done %
      adlmb  % address of first directory list master block %
      );
   LOCAL
      adlfb; % address of a directory list file block %
   LOCAL STRING
      filelink[200], % temp string for file link this file %
      pcname[200]    % temp string for pc name or file name %
      ;
   REF adlmb, adlfb;

   % RETURNS %
      % This routine returns the address of a directory list file
      block (obtained via dlgtblk and described by the record
      dlflbk) or FALSE if it fails because of running out of space %

   % get a directory list file block %
      IF NOT (&adlfb _ dlgtblk( &adlmb, dlfbl)) THEN RETURN( FALSE
      );
   % get fdb for the file (and account if neccessary) %
      IF NOT (adlfb.dlfbff _ dlgtblk( &adlmb, 25B)) THEN
         RETURN( FALSE );
      !gtfdb( fjfn, 25000000B, adlfb.dlfbff);
      IF info.dliacc THEN dlbgac(fjfn, adlfb.dlfbff, &adlmb);
   % get fdb for pc if neccessary (and account if neccessary)%
      IF pcjfn THEN
         BEGIN
         IF NOT (adlfb.dlfbpf _ dlgtblk( &adlmb, 25B)) THEN
            RETURN( FALSE );
         !gtfdb( pcjfn, 25000000B, adlfb.dlfbpf);
         IF info.dliacc THEN dlbgac(pcjfn, adlfb.dlfbpf, &adlmb);
         END;
   % build the file link for this file %
      jfnflink( fjfn, $filelink, 011110B6+1);
   % build string for second pass ( and set bits in dlfb %
      dlblds( fjfn, pcjfn, $pcname, &adlfb);
   % get the group key for this file %
      adlfb.dlfbgk _ dlgetg( gropk, adlfb.dlfbff, pcjfn,
      adlfb.dlfbpf);
   % get the sort key for this file %
      adlfb.dlfbsk _ dlgets( sortk, adlfb.dlfbff, pcjfn,
      adlfb.dlfbpf);
   % get directory list storage for the file link %
      IF (adlfb.dlfbal _ dlgtblk( &adlmb, (filelink.L+4)/5+1 )) THEN
         BEGIN
         [adlfb.dlfbal].M _ filelink.L;
         % place filelink in directory list storage %
            *[adlfb.dlfbal]* _ *filelink*;
         END
      ELSE RETURN( FALSE );
```

```
% get directory list storage for second pass string if needed %
   IF pcname.L THEN
      IF (adlfb.dlfbap _ dlgtblk( &adlmb, (pcname.L+4)/5+1 ))
      THEN
         BEGIN
         [adlfb.dlfbap].M _ pcname.L;
         % place second pass string in directory list storage %
            *[adlfb.dlfbap]* _ *pcname*;
         END
      ELSE RETURN( FALSE );
   IF adlfb.dlfbfl.dlstpc THEN adlfb.dlfbap _ adlfb.dlfbal;
% all done, so return address of the directory list file block %
   RETURN( &adlfb );

END.

%  %
```

```
(dlbldai)    % build additional line 1 information string %          5F
   PROCEDURE
      (fjfn, % jfn for the file %
      pcjfn, % jfn of the pc if it exists %
      jfnfdb,        % address of fdb for the file %
      pcfdb, % address of the fdb for the pc %
      string % address of string to receive the information %
      );
   LOCAL STRING
      locstr[100];   % temp string %
   REF jfnfdb, pcfdb, string;

   % get scratch or temporary info %
      IF jfnfdb[$fdbctl].fdbtmp THEN
         IF jfnfdb[$fdbver].LH >= 100000 THEN
            *string* _ *string*, "  Scratch"
         ELSE *string* _ *string*, "  Temporary";
   % get ephemeral status %
      IF jfnfdb[$fdbctl].fdbeph THEN
         *string* _ *string*, "  Ephemeron";
   % get deletion status %
      IF jfnfdb[$fdbctl].fdbdel THEN
         *string* _ *string*, "  Deleted";
   % now handle the pc %
      IF pcjfn THEN
         BEGIN
         lockid( $locstr, jfnfdb[$fdbusw].lkdirnum,
            jfnfdb[$fdbusw].lkinit );
         *string* _ *string*, "  [", *locstr*, SP;
         % get scratch or temporary info %
            IF pcfdb[$fdbctl].fdbtmp THEN
               IF pcfdb[$fdbver].LH >= 100000 THEN
                  *string* _ *string*, "Scratch "
               ELSE *string* _ *string*, "Temporary ";
         % get ephemeral status %
            IF pcfdb[$fdbctl].fdbeph THEN
               *string* _ *string*, "Ephemeron ";
         % get deletion status %
            IF pcfdb[$fdbctl].fdbdel THEN
               *string* _ *string*, "Deleted";
         *string* _ *string*, '];
         END;
   % done so return %
      RETURN;

   END.

   %  %
```

```
(dlbldi)     % build information string for this file %                    5G
   PROCEDURE
      (info, % dlinfo record for what is to be listed %
      pcjfn, % TRUE if pc exists %
      jfnfdb,          % address of fdb for the file %
      pcfdb, % address of the fdb for the pc %
      lead,  % address of strng to precede each line %
      string % address of string to receive the information %
      );
   REF jfnfdb, pcfdb, lead, string;

   % quick return if nothing requested %
      IF NOT info THEN RETURN;
   % get the account of this file %
      IF info.dliacc THEN
         BEGIN
         *string* _ *string*, *lead*, "Account: ";
         CASE jfnfdb[$fdbact] OF
            = -1: *string* _ *string*, "Can't Get File Account";
            >  0: *string* _ *string*, STRING( jfnfdb[$fdbact] );
            <  0: *string* _ *string*, *[jfnfdb[$fdbact].RH]*;
            ENDCASE;
         IF pcjfn THEN
            BEGIN
            *string* _ *string*, "   [";
            CASE pcfdb[$fdbact] OF
               = -1: *string* _ *string*, "Can't Get File Account";
               >  0: *string* _ *string*, STRING( pcfdb[$fdbact] );
               <  0: *string* _ *string*, *[pcfdb[$fdbact].RH]*;
               ENDCASE;
            *string* _ *string*, '];
            END;
         *string* _ *string*, CR, LF;
         END;
   % get the archive status of this file %
      IF info.dliars THEN
         BEGIN
         *string* _ *string*, *lead*, "Archive Status: ";
         dlbgas( jfnfdb[$fdbbkf].flarfl, &string);
         IF pcjfn THEN
            BEGIN
            *string* _ *string*, CR, LF, *lead*, "   [";
            dlbgas( pcfdb[$fdbbkf].flarfl, &string);
            *string* _ *string*, '];
            END;
         *string* _ *string*, CR, LF;
         END;
   % get the archive tape numbers of this file %
      IF info.dliart THEN
         BEGIN
         *string* _ *string*, *lead*, "Archive Tapes: ";
         IF NOT jfnfdb[$fdbart] THEN
            IF jfnfdb[$fdbbkf].flarfl.fdbaar THEN
               *string* _ *string*, "Not Available"
            ELSE *string* _ *string*, "Not Archived"
         ELSE
```

```
            *string* _ *string*, STRING( jfnfdb[$fdbart].flfsat), "
            and ", STRING( jfnfdb[$fdbart].flsnat);
         IF pcjfn THEN
            BEGIN
            *string* _ *string*, "   [";
            IF NOT pcfdb[$fdbart] THEN
               IF pcfdb[$fdbbkf].flarfl.fdbaar THEN
                  *string* _ *string*, "Not Available"
               ELSE *string* _ *string*, "Not Archived"
            ELSE
               *string* _ *string*, STRING( pcfdb[$fdbart].flfsat),
               " and ", STRING( pcfdb[$fdbart].flsnat);
            *string* _ *string*, '];
            END;
         *string* _ *string*, CR, LF;
         END;
   % get the dump tape number of this file %
     IF info.dlidmt THEN
         BEGIN
         *string* _ *string*, *lead*, "Dump Tape: ";
         IF jfnfdb[$fdbbkf].fldmpt THEN
            *string* _ *string*, STRING( jfnfdb[$fdbbkf].fldmpt )
         ELSE
            IF jfnfdb[$fdbtdm] THEN
               *string* _ *string*, "Not Available"
            ELSE
               *string* _ *string*, "Not Dumped";
         IF pcjfn THEN
            BEGIN
            *string* _ *string*, "   [";
            IF pcfdb[$fdbbkf].fldmpt THEN
               *string* _ *string*, STRING( pcfdb[$fdbbkf].fldmpt )
            ELSE
               IF pcfdb[$fdbtdm] THEN
                  *string* _ *string*, "Not Available"
               ELSE
                  *string* _ *string*, "Not Dumped";
            *string* _ *string*, ']];
            END;
         *string* _ *string*, CR, LF;
         END;
   % get the number of versions to keep of this file %
     IF info.dlidfr THEN
         BEGIN
         *string* _ *string*, *lead*, "No. Versions To Keep: ",
            STRING( jfnfdb[$fdbbyv].fldfr );
         IF pcjfn THEN
            *string* _ *string*,
            "   [", STRING( pcfdb[$fdbbyv].fldfr ), ']];
         *string* _ *string*, CR, LF;
         END;
   % get the last writer of this file %
     IF info.dlilwr THEN
         BEGIN
         *string* _ *string*, *lead*, "Last Writer: ";
         dlbglw( jfnfdb[$fdbuse].LH, &string);
```

```
            IF pcjfn THEN
               BEGIN
               *string* _ *string*, "   [";
               dlbglw( pcfdb[$fdbuse].LH, &string);
               *string* _ *string*, '];
               END;
            *string* _ *string*, CR, LF;
            END;
      % get the length and bytesize of this file %
         IF info.dlibyt THEN
            BEGIN
            *string* _ *string*, *lead*, "Length (and Bytesize): ",
               STRING( jfnfdb[$fdbsiz] ),
               '(, STRING( jfnfdb[$fdbbyv].flbyts ), ');
            IF pcjfn THEN
               *string* _ *string*, "   [",
               STRING( pcfdb[$fdbsiz] ),
               '(, STRING( pcfdb[$fdbbyv].flbyts ), ")]";
            *string* _ *string*, CR, LF;
            END;
      % get the number of accesses of this file %
         IF info.dlinrw THEN
            BEGIN
            *string* _ *string*, *lead*,
               "No. of Accesses (reads + writes): ",
               STRING( jfnfdb[$fdbcnt].flnmrd + jfnfdb[$fdbcnt].flnmwr
               ),
               " (", STRING( jfnfdb[$fdbcnt].flnmrd ), " + ",
               STRING( jfnfdb[$fdbcnt].flnmwr ), ');
            IF pcjfn THEN
               *string* _ *string*, "   [",
               STRING( pcfdb[$fdbcnt].flnmrd + pcfdb[$fdbcnt].flnmwr ),
               " (", STRING( pcfdb[$fdbcnt].flnmrd ), " + ",
               STRING( pcfdb[$fdbcnt].flnmwr ), ")]";
            *string* _ *string*, CR, LF;
            END;
      % get miscellaneous information for this file %
         IF info.dlimis THEN
            BEGIN
            *string* _ *string*, *lead*, "Misc. Info.: ";
            IF jfnfdb[$fdbctl].fdblng .V jfnfdb[$fdbctl].fdbprm THEN
               BEGIN
               IF jfnfdb[$fdbctl].fdblng THEN
                  *string* _ *string*, "Long File  ";
               IF jfnfdb[$fdbctl].fdbprm THEN
                  *string* _ *string*, "Permanent File";
               END
            ELSE *string* _ *string*, "None";
            IF pcjfn THEN
               BEGIN
               *string* _ *string*, "   [";
               IF jfnfdb[$fdbctl].fdblng .V jfntdb[$fdbctl].fdbprm THEN
                  BEGIN
                  IF jfnfdb[$fdbctl].fdblng THEN
                     *string* _ *string*, "Long File  ";
                  IF jfnfdb[$fdbctl].fdbprm THEN
                  *string* _ *string*, "Permanent File";
                END
             ELSE *string* _ *string*, "None";
             *string* _ *string*, "];
             END;
          *string* _ *string*, CR, LF;
          END;
    % get the protection of this file %
       IF info.dliprt THEN
          BEGIN
          *string* _ *string*, *lead*, "Protection: ";
          IF jfnfdb[$fdbprt].LH # 500000B THEN
             *string* _ *string*,
                "(, STRING( jfnfdb[$fdbprt], 8), ") - Fancy
                Protection"
          ELSE
             BEGIN
             *string* _ *string*,
                "(, STRING( jfnfdb[$fdbprt].RH, 8), "), CR, LF,
                   *lead*, "  Self ";
                dlbgpr( jfnfdb[$fdbprt].flprse, &string );
                *string* _ *string*, CR, LF, *lead*, "  Group ";
                dlbgpr( jfnfdb[$fdbprt].flprgr, &string );
                *string* _ *string*, CR, LF, *lead*, "  Public ";
                dlbgpr( jfnfdb[$fdbprt].flprpu, &string );
             END;
          IF pcjfn THEN
             BEGIN
             *string* _ *string*, CR, LF, *lead*, "   [ ";
             IF pcfdb[$fdbprt].LH # 500000B THEN
                *string* _ *string*,
                   "(, STRING( pcfdb[$fdbprt], 8), ") - Fancy
                   Protection ]"
             ELSE
                BEGIN
                *string* _ *string*,
                   "(, STRING( pcfdb[$fdbprt].RH, 8), "), CR, LF,
                      *lead*, "    Self ";
                   dlbgpr( pcfdb[$fdbprt].flprse, &string );
                   *string* _ *string*, CR, LF, *lead*, "    Group ";
                   dlbgpr( pcfdb[$fdbprt].flprgr, &string );
                   *string* _ *string*, CR, LF, *lead*, "    Public
                   ";
                   dlbgpr( pcfdb[$fdbprt].flprpu, &string );
                   *string* _ *string*, " ]";
                END;
             END;
          *string* _ *string*, CR, LF;
          END;
    % get the size of this file %
       IF info.dlisiz THEN
          BEGIN
          *string* _ *string*, *lead*, "Size in Pages: ",
             STRING( jfnfdb[$fdbbyv].flpgsz );
          IF pcjfn THEN
             *string* _ *string*,
                "   [", STRING( pcfdb[$fdbbyv].flpgsz ), "];
           *string* _ *string*, CR, LF;
           END;
     % get [time and] date of archiving of this file %
        IF info.dlitar THEN
           BEGIN
           *string* _ *string*, *lead*, "Archived: ";
           IF jfnfdb[$fdbtsa] THEN
              dlbgtd( info.dlitar, jfnfdb[$fdbtsa], &string)
           ELSE
              IF jfnfdb[$fdbbkf].flarfl.fdbaar THEN
                 *string* _ *string*, "Not Available"
              ELSE
                 *string* _ *string*, "Not Archived";
           IF pcjfn THEN
              BEGIN
              *string* _ *string*, "   [";
              IF pcfdb[$fdbtsa] THEN
                 dlbgtd( info.dlitar, pcfdb[$fdbtsa], &string)
              ELSE
                 IF pcfdb[$fdbbkf].flarfl.fdbaar THEN
                    *string* _ *string*, "Not Available"
                 ELSE
                    *string* _ *string*, "Not Archived";
              *string* _ *string*, "];
              END;
           *string* _ *string*, CR, LF;
           END;
     % get [time and] date of creation of this file %
        IF info.dlitcr THEN
           BEGIN
           *string* _ *string*, *lead*, "Created: ";
           IF jfnfdb[$fdbcrv] THEN
              dlbgtd( info.dlitcr, jfnfdb[$fdbcrv], &string)
           ELSE *string* _ *string*, "Not Available";
           IF pcjfn THEN
              BEGIN
              *string* _ *string*, "   [";
              IF pcfdb[$fdbcrv] THEN
                 dlbgtd( info.dlitcr, pcfdb[$fdbcrv], &string)
              ELSE *string* _ *string*, "Not Available";
              *string* _ *string*, "];
              END;
           *string* _ *string*, CR, LF;
           END;
     % get [time and] date of dumping of this file %
        IF info.dlitdm THEN
           BEGIN
           *string* _ *string*, *lead*, "Dumped: ";
           IF jfnfdb[$fdbtdm] THEN
              dlbgtd( info.dlitdm, jfnfdb[$fdbtdm], &string)
           ELSE *string* _ *string*, "Not Dumped";
           IF pcjfn THEN
              BEGIN
              *string* _ *string*, "   [";
              IF pcfdb[$fdbtdm] THEN
```

```
                dlbgtd( info.dlitdm, pcfdb[$fdbtdm], &string)
             ELSE *string* _ *string*, "Not Dumped";
             *string* _ *string*, ']; 
             END;
          *string* _ *string*, CR, LF;
          END;
    % get [time and] date of original version creation of this file %
       IF info.dlitov THEN
          BEGIN
          *string* _ *string*, *lead*, "Original Version Created: ";
          IF jfnfdb[$fdbcre] THEN
             dlbgtd( info.dlitov, jfnfdb[$fdbcre], &string)
          ELSE *string* _ *string*, "Not Available";
          IF pcjfn THEN
             BEGIN
             *string* _ *string*, "   [";
             IF pcfdb[$fdbcre] THEN
                dlbgtd( info.dlitov, pcfdb[$fdbcre], &string)
             ELSE *string* _ *string*, "Not Available";
             *string* _ *string*, '];
             END;
          *string* _ *string*, CR, LF;
          END;
    % get [time and] date of last reading of this file %
       IF info.dlitrd THEN
          BEGIN
          *string* _ *string*, *lead*, "Last Read: ";
          IF jfnfdb[$fdbref] THEN
             dlbgtd( info.dlitrd, jfnfdb[$fdbref], &string)
          ELSE *string* _ *string*, "Never Read";
          IF pcjfn THEN
             BEGIN
             *string* _ *string*, "   [";
             IF pcfdb[$fdbref] THEN
                dlbgtd( info.dlitrd, pcfdb[$fdbref], &string)
             ELSE *string* _ *string*, "Never Read";
             *string* _ *string*, '];
             END;
          *string* _ *string*, CR, LF;
          END;
    % get [time and] date of last writing of this file %
       IF info.dlitwr THEN
          BEGIN
          *string* _ *string*, *lead*, "Last Written: ";
          IF jfnfdb[$fdbwrt] THEN
             dlbgtd( info.dlitwr, jfnfdb[$fdbwrt], &string)
          ELSE *string* _ *string*, "Never Written";
          IF pcjfn THEN
             BEGIN
             *string* _ *string*, "   [";
             IF pcfdb[$fdbwrt] THEN
                dlbgtd( info.dlitwr, pcfdb[$fdbwrt], &string)
             ELSE *string* _ *string*, "Never Written";
             *string* _ *string*, '];
             END;
          *string* _ *string*, CR, LF;
```

```
        END;
    % done finally so return %
      RETURN;

    END.

    %  %
```

```
(dlgetg)      % return the group key for this file %                5H
   PROCEDURE
      (gropk,           % dlgrp record %
      jfnfdb,           % address of the file fdb %
      pcjfn, % jfn for the pc if it exists %
      pcfdb  % address of the pc fdb %
      );
   REF jfnfdb, pcfdb;

   % recurse if reverse grouping %
      IF (gropk.dlgrvr := FALSE) THEN
         RETURN( dlgetg(gropk, &jfnfdb, pcjfn, &pcfdb) .X -1 );
   % see if any grouping %
      IF NOT gropk THEN RETURN( 35M );        % no grouping %
   IF gropk.dlgacc THEN         % group by account %
      IF jfnfdb[$fdbact] < 0 THEN RETURN( jfnfdb[$fdbact].RH )
      ELSE RETURN( jfnfdb[$fdbact] );
   IF gropk.dlgars THEN         % group by archive status %
      RETURN( jfnfdb[$fdbbkf].flarfl );
   IF gropk.dlgdar THEN         % group by archive date %
      RETURN( jfnfdb[$fdbtsa].LH );
   IF gropk.dlgart THEN         % group by archive tapes %
      RETURN( jfnfdb[$fdbart].flsnat );
   IF gropk.dlgbyt THEN         % group by bytesize %
      RETURN( jfnfdb[$fdbbyv].flbyts );
   IF gropk.dlgdcr THEN         % group by creation date %
      RETURN( jfnfdb[$fdbcrv].LH );
   IF gropk.dlgdlt THEN         % group by deletion status %
      RETURN( jfnfdb[$fdbctl].fdbdel );
   IF gropk.dlgddm THEN         % group by dump date %
      RETURN( jfnfdb[$fdbtdm].LH );
   IF gropk.dlgdmt THEN         % group by dump tape %
      RETURN( jfnfdb[$fdbbkf].fldmpt );
   IF gropk.dlgdfr THEN         % group by no. of versions to keep %
      RETURN( jfnfdb[$fdbbyv].fldfr );
   IF gropk.dlglwr THEN         % group by last writer %
      IF pcjfn THEN RETURN( pcfdb[$fdbuse].LH )
      ELSE RETURN( jfnfdb[$fdbuse].LH );
   IF gropk.dlgdov THEN         % group by orig. vers. creation date %
      RETURN( jfnfdb[$fdbcre].LH );
   IF gropk.dlgprt THEN         % group by protection %
      RETURN( jfnfdb[$fdbprt] );
   IF gropk.dlgdrd THEN         % group by last read date %
      IF pcjfn THEN
         RETURN( MAX( jfnfdb[$fdbref].LH, pcfdb[$fdbref].LH) )
      ELSE RETURN( jfnfdb[$fdbref].LH );
   IF gropk.dlgdwr THEN         % group by last write date %
      IF pcjfn THEN
         RETURN( MAX( jfnfdb[$fdbwrt].LH, pcfdb[$fdbwrt].LH) )
      ELSE RETURN( jfnfdb[$fdbwrt].LH );
   RETURN( 35M );               % no grouping (or bad record) %

   END.

   %  %
```

```
(dlgets)     % return the sort key for this file %                     5I
   PROCEDURE
      (sortk,        % dlsort record %
      jfnfdb,        % address of the file fdb %
      pcjfn, % jfn for the pc if it exists %
      pcfdb  % address of the pc fdb %
      );
   REF jfnfdb, pcfdb;

   % find out if any sorting %
      IF NOT sortk THEN RETURN( 35M );        % no sorting %
   IF sortk.dlsacc THEN        % sort by accounts %
      IF jfnfdb[$fdbact] < 0 THEN
         RETURN( jfnfdb[$fdbact].RH )
      ELSE RETURN( jfnfdb[$fdbact] );
   IF sortk.dlsart THEN        % sort by archive tapes %
      RETURN( jfnfdb[$fdbart].flsnat );
   IF sortk.dlstar THEN        % sort by archive time and date %
      RETURN( jfnfdb[$fdbtsa] );
   IF sortk.dlsbyt THEN        % sort by bytesize %
      RETURN( jfnfdb[$fdbbyv].flbyts );
   IF sortk.dlstcr THEN        % sort by time and date of creation %
      RETURN( jfnfdb[$fdbcrv] );
   IF sortk.dlsdlt THEN        % sort by deletion status %
      RETURN( jfnfdb[$fdbctl].fdbdel );
   IF sortk.dlstdm THEN        % sort by time and date of dump %
      RETURN( jfnfdb[$fdbtdm] );
   IF sortk.dlsdmt THEN        % sort by dump tape %
      RETURN( jfnfdb[$fdbbkf].fldmpt );
   IF sortk.dlsdfr THEN        % sort by file retention specs %
      RETURN( jfnfdb[$fdbbyv].fldfr );
   IF sortk.dlslwr THEN        % sort by last writer %
      IF pcjfn THEN RETURN( pcfdb[$fdbuse].LH )
      ELSE RETURN( jfnfdb[$fdbuse].LH );
   IF sortk.dislen THEN        % sort by length in bytes %
      IF pcjfn THEN RETURN( jfnfdb[$fdbsiz] + pcfdb[$fdbsiz] )
      ELSE RETURN( jfnfdb[$fdbsiz] );
   IF sortk.dlsnac THEN        % sort by number of accesses %
      IF pcjfn THEN
         RETURN(
            jfnfdb[$fdbcnt].flnmrd + jfnfdb[$fdbcnt].flnmwr +
            pcfdb[$fdbcnt].flnmrd + pcfdb[$fdbcnt].flnmwr )
      ELSE RETURN( jfnfdb[$fdbcnt].flnmrd + jfnfdb[$fdbcnt].flnmwr
      );
   IF sortk.dlsnrd THEN        % sort by number of reads %
      IF pcjfn THEN
         RETURN( jfnfdb[$fdbcnt].flnmrd + pcfdb[$fdbcnt].flnmrd )
      ELSE RETURN( jfnfdb[$fdbcnt].flnmrd );
   IF sortk.dlsnwr THEN        % sort by number of writes %
      IF pcjfn THEN
         RETURN( jfnfdb[$fdbcnt].flnmwr + pcfdb[$fdbcnt].flnmwr )
      ELSE RETURN( jfnfdb[$fdbcnt].flnmwr );
   IF sortk.dlstov THEN        % sort by orig. ver. creation t & d %
      RETURN( jfnfdb[$fdbcre] );
   IF sortk.dlstrd THEN        % sort by last read time and date %
      IF pcjfn THEN RETURN( MAX( jfnfdb[$fdbref], pcfdb[$fdbref]) )
      ELSE RETURN( jfnfdb[$fdbref] );
   IF sortk.dlssiz THEN      % sort by size in pages %
      IF pcjfn THEN
         RETURN( jfnfdb[$fdbbyv].flpgsz + pctdb[$fdbbyv].flpgsz )
      ELSE RETURN( jfnfdb[$fdbbyv].flpgsz );
   IF sortk.dlstwr THEN       % sort by last write time and date %
      IF pcjfn THEN RETURN( MAX( jfnfdb[$fdbwrt], pcfdb[$fdbwrt]) )
      ELSE RETURN( jfnfdb[$fdbwrt] );
   RETURN( 35M );    % bad input, thus no sort %

   END.

   % %
```

```
(dlinfb)     % insert a dlfb in chain %                             5J
   PROCEDURE
      (chns, % address of chain start %
      adlmb, % address of directory list master block %
      adlfb, % address of dlfb %
      sortk  % sort specification %
      );
   LOCAL
      gptr,  % current group block pointer %
      lptr,  % current position pointer %
      tptr   % temporary pointer %
      ;
   REF chns, adlmb, adlfb, lptr, tptr, gptr;

   % find the right directory group block %
      IF NOT chns THEN        % no group chain exists yet %
         BEGIN
         % create group block for the group %
         % if no room return since we can't do anything %
            IF NOT ([&chns] _ dlgtblk(&adlmb, dlgbl)) THEN
               RETURN( FALSE );
         % make this the group block for this group %
            [chns].dlgbgk _ adlfb.dlfbgk;
         % set up current pointer %
            &lptr _ [&chns];
         END
      ELSE
         CASE adlfb.dlfbgk OF
            < [chns].dlgbgk: % this group fits in front of first
            group %
               BEGIN
               % get new group block (if can't then return) %
                  IF NOT (&tptr _ dlgtblk(&adlmb, dlgbl)) THEN
                     RETURN( FALSE );
               % set up this block to be for this group %
                  tptr.dlgbgk _ adlfb.dlfbgk;
               % chain in new block & fix up start of chain pointer
               %
                  tptr.dlgbnb _ [&chns];
                  [chns].dlgbpb _ &tptr;
                  [&chns] _ &lptr _ &tptr;
               END;
            = [chns].dlgbgk: % this file is in first group %
               &lptr _ [&chns];
            > [chns].dlgbgk: % this group fits after first group %
               BEGIN
               % set up current pointer to loop through chain %
                  &lptr _ [&chns];
               LOOP
                  IF NOT lptr.dlgbnb THEN
                     % chain end, add new group (return if no space)
                     %
                     BEGIN
                     IF NOT (&tptr _ dlgtblk(&adlmb, dlgbl)) THEN
                        RETURN( FALSE );
                     tptr.dlgbgk _ adlfb.dlfbgk;
```

```
                        tptr.dlgbpb _ &lptr;
                        lptr.dlgbnb _ &tptr;
                        &lptr _ &tptr;
                        EXIT LOOP;
                        END
                    ELSE
                        % not chain end, look at next block %
                        BEGIN
                        &lptr _ lptr.dlgbnb;
                        CASE adlfb.dlfbgk OF
                            < lptr.dlgbgk: % before this one (after
                            last) %
                                BEGIN
                                IF NOT (&tptr _ dlgtblk(&adlmb, dlgbl))
                                THEN
                                    RETURN( FALSE );
                                tptr.dlgbgk _ adlfb.dlfbgk;
                                tptr.dlgbnb _ &lptr;
                                tptr.dlgbpb _ lptr.dlgbpb;
                                lptr.dlgbpb _ [lptr.dlgbpb].dlgbnb _
                                &tptr;
                                &lptr _ &tptr;
                                EXIT LOOP;
                                END;
                            = lptr.dlgbgk: % this is the one! %
                                EXIT LOOP;
                            > lptr.dlgbgk: % greater, continue search %
                                REPEAT LOOP;
                            ENDCASE;     % can't possibly happen %
                        END;
                END;
            ENDCASE; % can't possibly happen %
    % by now lptr points to proper group block %
    % find proper sort place within this group %
        IF NOT lptr.dlgbsc THEN
            BEGIN               % no entries yet this group %
            % make this the first one (and we're done) %
                lptr.dlgbsc _ lptr.dlgbnu _ &adlfb;
            RETURN( TRUE );
            END
        ELSE
            CASE adlfb.dlfbsk OF
                >= [lptr.dlgbnu].dlfbsk: % goes after last entry %
                    BEGIN
                    adlfb.dlfbpb _ lptr.dlgbnu;
                    lptr.dlgbnu _ [lptr.dlgbnu].dlfbnb _ &adlfb;
                    RETURN( TRUE );
                    END;
                ENDCASE                 % goes before last entry %
                    BEGIN
                    % save pointer to this group block %
                        &gptr _ &lptr;
                    % move current pointer to last entry %
                        &lptr _ lptr.dlgbnu;
                    LOOP
                        IF NOT lptr.dlfbpb THEN    % hit end of chain %
```

```
                         BEGIN
                         adlfb.dlfbnb _ &lptr;
                         lptr.dlfbpb _ &adlfb;
                         gptr.dlgbsc _ &adlfb;
                         RETURN( TRUE );
                         END
                      ELSE                                           % not chain end
                      yet %
                         BEGIN
                         % step over to next entry and see what happens
                         %
                            &lptr _ lptr.dlfbpb;
                            CASE adlfb.dlfbsk OF
                               >= lptr.dlfbsk:   % after this one %
                                  BEGIN
                                  adlfb.dlfbpb _ &lptr;
                                  adlfb.dlfbnb _ lptr.dlfbnb;
                                  lptr.dlfbnb _
                                     [lptr.dlfbnb].dlfbpb _ &adlfb;
                                  RETURN( TRUE );
                                  END;
                               ENDCASE REPEAT LOOP; % before this one,
                               loop %
                         END;
                   END;

   END.

   %  %
```

```
(dlsnps)      % directory command 2 pass - get rid of redundant PCs %
                                                                    5K
   PROCEDURE
      (nlschn,        % address 1st dlfb of nls file with a pc %
      pcchn  % address 1st dlfb of pc file %
      );
   LOCAL
      tpcchn;         % loop variable %
   REF nlschn, pcchn, tpcchn;

   % special quick exit %
      IF NOT &pcchn THEN RETURN;
   % loop thru nls chain %
      WHILE &nlschn # 0 DO
         BEGIN
         &tpcchn _ &pcchn;
         WHILE &tpcchn # 0 DO
            BEGIN
            IF NOT tpcchn.dlfbfl.dlstig THEN
               IF *[nlschn.dlfbap]* = *[tpcchn.dlfbap]* THEN
                  BEGIN
                  tpcchn.dlfbfl.dlstig _ TRUE;
                  EXIT LOOP;
                  END;
            &tpcchn _ tpcchn.dlfbcp;
            END;
         &nlschn _ nlschn.dlfbcp;
         END;
   RETURN;

   END.

   %  %
```

```
(dlckdl)      % check deletion status of this file %                  5L
   PROCEDURE
      (fjfn, % jfn of the file %
      delbit % request bits %
      );

   CASE delbit OF
      = 1:    % deleted only %
         BEGIN
         !gtfdb( fjfn, 1B6+1, $R3);
         IF R3.fdbdel THEN RETURN( TRUE )
         ELSE RETURN( FALSE );
         END;
      ENDCASE RETURN( TRUE );  % wouldn't have jfn if false %

   END.

   %  %
```

```
(dlblds)      % build string for second pass ( and set bits in dlfb %
                                                                    5M
   PROCEDURE
      (fjfn, % jfn of the file %
      pcjfn, % jfn of the pc if it exists %
      string,        % address of string to receive the information
      %
      adlfb  % address of the dlfb %
      );
   LOCAL STRING
      locstr[40];    % temp string %
   REF string, adlfb;

   % if no pc then see if this is a pc %
      IF NOT pcjfn THEN
         BEGIN                % find out if this is a pc %
         % get extension name %
            jfntostr( fjfn, $locstr, 000100B6);
         % now see if it is PC %
            IF *locstr* = "PC" THEN
               BEGIN % this is a pc %
               adlfb.dlfbfl.dlstpc _ TRUE;
               RETURN;
               END
            ELSE RETURN;     % not a PC %

         END;
   % has a pc so setup and return %
      adlfb.dlfbfl.dlstnl _ TRUE;
      jfnflink( pcjfn, $string, 011110B6+1);
      RETURN;

   END.

   %  %
```

```
(dlbgac)      % get account string or number for a directory list % 5N
   PROCEDURE
      (fjfn, % jfn of the file %
      fdb,   % address of file's fdb %
      adlmb  % address of dlmb %
      );
   LOCAL
      index; % temp for concatenating strings %
   LOCAL STRING
      tstring[40]; - % temp to hold alphanumeric account strings %
   REF fdb, adlmb;

   tstring.L _ tstring.M;
   !gactf( fjfn, chbmty+$tstring);
      GOTO gerr;              % something really screwed up %
      GOTO acisstr;  % account is string %
      GOTO acisnum;  % account is number %
   (gerr):                                                          5N8
      fdb[$fdbact] _ -1;
      RETURN;
   (acisstr):                                                       5N9
      FOR index _ 1 UP UNTIL > tstring.L DO
         IF *tstring*[index] # 0 THEN NULL
         ELSE
            BEGIN
            tstring.L _ index _ index - 1;
            EXIT LOOP;
            END;
      IF index <= 0 THEN
         BEGIN
         fdb[$fdbact] _ -1;
         RETURN;
         END;
      IF NOT (fdb[$fdbact] _ dlgtblk( &adlmb, (index+4)/5+1) ) THEN
         fdb[$fdbact] _ -1
      ELSE
         BEGIN
         [fdb[$fdbact]].M _ index;
         *[fdb[$fdbact]]* _ *tstring*;
         fdb[$fdbact] _ fdb[$fdbact] .V 5B11;
         END;
      RETURN;
   (acisnum):                                                       5N10
      fdb[$fdbact] _ R2.RH;
      RETURN;

   END.

   %  %
```

```
(dlbgtd)      % append [time and] date string to passed string %     50
   PROCEDURE
      (param,          % field indicating date or time and date
      request %
      time,  % time and date %
      string % string to be appended to %
      );
   LOCAL STRING
      locstr[40];     % temp string %
   REF string;

   % catch any errors %
      INVOKE (sigstr, RETURN);
   % now convert time appropriately %
      CASE param OF
         =1: % date only %
            datfrmt( time, 401B6, $locstr);
         ENDCASE      % time and date %
            datfrmt( time, 1B6, $locstr);
   % append the string and return %
      *string* _ *string*, *locstr*;
      DROP (sigstr);
      RETURN;

   (sigstr) CATCHPHRASE;                                              509
      BEGIN
      DISABLE (sigstr);
      *string* _ *string*, "Can't Convert [Time and] Date";
      TERMINATE;
      END;


   END.

   %  %
```

```
(dlbgpr)     % get protection string for individuall field %          5P
   PROCEDURE
      (prot, % protection bits for this field %
      string % address of string to be appended to %
      );
   REF string;

   % print the numeric protection first %
      *string* _ *string*, "(, STRING( prot, 8), ") - ";
   % now interpret the bits %
      IF NOT prot THEN
         *string* _ *string*, "No Access "
      ELSE
         BEGIN
         IF (prot.flprre := FALSE) THEN
            IF prot THEN *string* _ *string*, "Read, "
            ELSE *string* _ *string*, "Read ";
         IF (prot.flprwr := FALSE) THEN
            IF prot THEN *string* _ *string*, "Write, "
            ELSE *string* _ *string*, "Write ";
         IF (prot.flprex := FALSE) THEN
            IF prot THEN *string* _ *string*, "Execute, "
            ELSE *string* _ *string*, "Execute ";
         IF (prot.flprap := FALSE) THEN
            IF prot THEN *string* _ *string*, "Append, "
            ELSE *string* _ *string*, "Append ";
         IF prot.flprli THEN *string* _ *string*, "List ";
         END;
   % finish up and return %
      *string* _ *string*, "Allowed";
      RETURN;

   END.

   %  %
```

```
(dlbglw)      % get last writer string %                              5Q
   PROCEDURE
      (dirno,        % directory number of last writer %
      string % string to get info %
      );
   LOCAL STRING
      dirname[40];   % temp string to get directory name %
   REF string;

   % catch any errors %
      INVOKE (sigstr, RETURN);
   % check for null field %
      IF NOT dirno THEN
         BEGIN
         *string* _ *string*, "SYSTEM";
         DROP (sigstr);
         RETURN;
         END;
   % now get the string and add it to passed string %
      gdname(dirno, $dirname);
      *string* _ *string*, *dirname*;
   % done so return %
      DROP (sigstr);
      RETURN;

   (sigstr) CATCHPHRASE;                                             5Q10
      BEGIN
      DISABLE (sigstr);
      *string* _ *string*, "can't convert directory number";
      TERMINATE;
      END;


   END.

   %  %
```

```
(dibgas)     % get archive status string %                                  58
   PROCEDURE
      (flags,          % flags indicating archive status %
      string % address of string to get archive status %
      );
   REF string;

   IF NOT
      (flags.fdbaar .V flags.fdbadl .V flags.fdbnar .V flags.fdbarc)
      THEN BEGIN
      *string* _ *string*, "None";
      RETURN;
      END;
   IF flags.fdbaar THEN
      *string* _ *string*, "Already Archived  ";
   IF flags.fdbadl THEN
      *string* _ *string*, "Don't Delete After Archiving  ";
   IF flags.fdbnar THEN
      *string* _ *string*, "Archive Not Allowed  ";
   IF flags.fdbarc THEN
      *string* _ *string*, "Archive Pending";
   RETURN;
   END.

FINISH of execfl
```

```
BLP,    16-Aug-78 00:09 T=1, L=1,    < NINE, INDEX-FILMNP.NLS;6, >   1

(badfil)        <nine, filmnp, 02238>          LOCAL          12B1
(copfgrp)       <nine, filmnp, 0583>           PROCEDURE      10B3
(copgrp)        <nine, filmnp, 0545>           LOCAL          10B2
(copit)         <nine, filmnp, 01154>          PROCEDURE      10E2
(copnam)        <nine, filmnp, 01667>          PROCEDURE      11C5
(copplist)      <nine, filmnp, 0847>           LOCAL          10C1
(copprop)       <nine, filmnp, 0900>           PROCEDURE      10D2
(creit)         <nine, filmnp, 01138>          PROCEDURE      10E1
(crenod)        <nine, filmnp, 0531>           PROCEDURE      10B1
(crepc)         <nine, filmnp, 02488>          PROCEDURE      13C1
(crepr2)        <nine, filmnp, 01311>          PROCEDURE      11A2
(creprop)       <nine, filmnp, 0894>           PROCEDURE      10D1
(delgrp)        <nine, filmnp, 0820>           LOCAL          10B6
(delit)         <nine, filmnp, 01208>          PROCEDURE      10E4
(delprop)       <nine, filmnp, 01072>          PROCEDURE      10D4
(fblk)          <nine, filmnp, 02684>          CATCHPHRASE    11F3W
(fchtxt)        <nine, filmnp, 0295>           LOCAL          9F10
(filhdr)        <nine, filmnp, 02266>          PROCEDURE      12D1
(freintree)     <nine, filmnp, 01443>          PROCEDURE      11A4
(freplist)      <nine, filmnp, 0869>           PROCEDURE      10C2
(freprop)       <nine, filmnp, 01229>          PROCEDURE      11A1
(frerng)        <nine, filmnp, 01886>          LOCAL          11E3
(frzblk)        <nine, filmnp, 02225>          LOCAL          12A2
(frzrtb)        <nine, filmnp, 02194>          LOCAL          12A1
(gcol)          <nine, filmnp, 02600>          LOCAL          11F3
(getall)        <nine, filmnp, 033>            LOCAL          9A3
(getbck)        <nine, filmnp, 097>            LOCAL          9A9
(getdblen)      <nine, filmnp, 0284>           LOCAL          9F8
(getend)        <nine, filmnp, 070>            LOCAL          9A7
(getfhd)        <nine, filmnp, 0135>           LOCAL          9B2
(getftl)        <nine, filmnp, 0129>           LOCAL          9B1
(gethed)        <nine, filmnp, 050>            LOCAL          9A5
(getint)        <nine, filmnp, 0249>           LOCAL          9F5
(getitree)      <nine, filmnp, 0181>           PROCEDURE      9E1
(getnam)        <nine, filmnp, 0147>           LOCAL          9B4
(getnmdl)       <nine, filmnp, 0211>           PROCEDURE      9F3
(getnmt)        <nine, filmnp, 0141>           LOCAL          9B3
(getnxt)        <nine, filmnp, 0107>           LOCAL          9A10
(getorf)        <nine, filmnp, 0205>           LOCAL          9F2
(getorg)        <nine, filmnp, 0191>           LOCAL          9F1
(getphed)       <nine, filmnp, 0229>           LOCAL          9F4
(getprd)        <nine, filmnp, 058>            LOCAL          9A6
(getpsdb)       <nine, filmnp, 0171>           PROCEDURE      9D1
(getptab)       <nine, filmnp, 01598>          PROCEDURE      11C2
(getsdb)        <nine, filmnp, 0160>           LOCAL          9C1
(getsid)        <nine, filmnp, 0153>           LOCAL          9B5
(getstsize)     <nine, filmnp, 0289>           LOCAL          9F9
(getsub)        <nine, filmnp, 026>            LOCAL          9A2
(getsuc)        <nine, filmnp, 019>            LOCAL          9A1
(gettim)        <nine, filmnp, 0279>           LOCAL          9F7
(getup)         <nine, filmnp, 042>            LOCAL          9A4
(getvnd)        <nine, filmnp, 078>            LOCAL          9A8
(goodrng)       <nine, filmnp, 01921>          PROCEDURE      11E4
(initdb)        <nine, filmnp, 02092>          PROCEDURE      11F2
(insd)          <nine, filmnp, 01492>          LOCAL          11B5
(insgrp)        <nine, filmnp, 01469>          LOCAL          11B3
```

```
(insitree)     <nine, filmnp, 01435>          LOCAL            11A3
(inss)         <nine, filmnp, 01479>          LOCAL            11B4
(levset)       <nine, filmnp, 01526>          LOCAL            11B7
(lnkprop)      <nine, filmnp, 01605>          PROCEDURE        11C3
(locprop)      <nine, filmnp, 01553>          PROCEDURE        11C1
(lodent)       <nine, filmnp, 0373>           LOCAL            10A1
(lodprop)      <nine, filmnp, 0409>           PROCEDURE        10A2
(lodrfb)       <nine, filmnp, 0433>           LOCAL            10A3
(movit)        <nine, filmnp, 01175>          PROCEDURE        10E3
(movprop)      <nine, filmnp, 0992>           PROCEDURE        10D3
(mvdifgrp)     <nine, filmnp, 0648>           LOCAL            10B4
(newdb)        <nine, filmnp, 01934>          LOCAL            11F1
(newrng)       <nine, filmnp, 01749>          LOCAL            11E1
(newsub)       <nine, filmnp, 01460>          LOCAL            11B2
(newsuc)       <nine, filmnp, 01451>          LOCAL            11B1
(nwrngb)       <nine, filmnp, 01844>          LOCAL            11E2
(remgrp)       <nine, filmnp, 0796>           LOCAL            10B5
(reprop)       <nine, filmnp, 01088>          PROCEDURE        10D5
(rlevset)      <nine, filmnp, 01510>          LOCAL            11B6
(rtnfl0)       <nine, filmnp, 01744>          LOCAL            11D1N
(sigcls)       <nine, filmnp, 0789>           CATCHPHRASE      10B4P
(sigrtn)       <nine, filmnp, 014>            CATCHPHRASE      7A
(stofhd)       <nine, filmnp, 0323>           LOCAL            9G4
(stofti)       <nine, filmnp, 0316>           LOCAL            9G3
(stoitree)     <nine, filmnp, 0365>           LOCAL            9G10
(stonam)       <nine, filmnp, 0344>           LOCAL            9G7
(stonmf)       <nine, filmnp, 0330>           LOCAL            9G5
(stoorg)       <nine, filmnp, 0358>           LOCAL            9G9
(stosdb)       <nine, filmnp, 0337>           LOCAL            9G6
(stosid)       <nine, filmnp, 0351>           LOCAL            9G8
(stosub)       <nine, filmnp, 0309>           LOCAL            9G2
(stosuc)       <nine, filmnp, 0302>           LOCAL            9G1
(trecop)       <nine, filmnp, 01702>          LOCAL            11D1
(trnbt7)       <nine, filmnp, 02730>          STRING           4A
(trnsint)      <nine, filmnp, 0262>           PROCEDURE        9F6
(unlnkprop)    <nine, filmnp, 01625>          PROCEDURE        11C4
(upctbl)       <nine, filmnp, 02244>          LOCAL            12C1
(wpiabt)       <nine, filmnp, 02390>          PROC             13A3
(wpifat)       <nine, filmnp, 02751>          PROC             13A4
(wrpi)         <nine, filmnp, 02307>          LOCAL            13A2
(wrpsproc)     <nine, filmnp, 02297>          PROCEDURE        13A1
```

```
< NINE, FILMNP.NLS;5, >, 3-May-78 22:13 BLP ;;;;
FILE filmnp  % <ARCSUBSYS>XL10  <RELNINE>filmnp %  % (arcsubsys,xL10,)
(RELNINE,filmnp.rel,) %
ALLOW!
%.....FILE MANIPULATION SUPPORT ROUTINES.....%
%Declarations%

   (trn5t7) STRING = " ABCDEFGHIJKLMNOPQRSTUVWXYZ23456";               4A
      %for 5- to 7-bit character translation%
   EXTERNAL wrtpsi;

 % File-global catchphrases %

   (sigrtn) CATCHPHRASE;                                               7A
      CASE SIGNALTYPE OF
         = aborttype:
            TERMINATE;   %RETURN (FALSE)%
         ENDCASE CONTINUE;

% Gets and stores %
   % Get --  Structural relations %
      (getsuc) %The stid for the successor field is returned.  If there
      is no successor, the stid of the up is returned.%
                                                                     9A1
         PROCEDURE (stid);
         LOCAL rngloc;
         lodent(stid, rngtyp : rngloc);
         IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
         stid.stpsid _ [rngloc].rsuc;
         RETURN (stid) END.

      (getsub) %The STID in the sub field is returned.%
                                                                     9A2
         PROCEDURE (stid);
         LOCAL rngloc;
         lodent(stid, rngtyp : rngloc);
         IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
         stid.stpsid _ [rngloc].rsub;
         RETURN (stid) END.

      (getail)                                                       9A3
         %Given an stid, this procedure returns the stid of the tail of
         the current plex%
         %-------------%
         PROCEDURE (stid);
         IF stid.stpsid # origin AND NOT getorf( stid ) THEN
            UNTIL getftl(stid) DO
            stid _ getsuc(stid);
         RETURN(stid);
         END.

      (getup)                                                        9A4
         %Given an stid, this routine returns the stid of the source of
         the original stid%
         %-------------%
         PROCEDURE(stid);
      IF stid.stpsid # origin AND NOT getorf( stid ) THEN
         stid _ getsuc(getail(stid));
      RETURN(stid);
      END.

   (gethed)                                                        9A5
      %Given an stid, this routine returns the head of the current
      plex; if it is passed the origin statement, it returns the
      origin%
      %-------------%
      PROCEDURE(stid);
      IF stid.stpsid # origin AND NOT getorf( stid ) THEN
         stid _ getsub(getup(stid));
      RETURN(stid);
      END.

   (getprd)                                                        9A6
      %Given an stid, this routine returns the predecessor; if the
      psid heads a plex, the stid itself is returned%
      %-------------%
      PROCEDURE (stid);
      LOCAL presid, %stid of predecessor%
         sucsid; %stid of successor of presid%
      IF getfhd(stid) THEN RETURN(stid);
      presid _ gethed(stid);
      UNTIL (sucsid _ getsuc(presid)) = stid DO
         presid _ sucsid;
      RETURN(presid);
      END.

   (getend)                                                        9A7
      %This procedure returns the end of the branch headed by the
      stid passed it.%
      %-------------%
      PROCEDURE(stid);
      UNTIL (stid := getsub(stid)) = stid
      DO stid _ getail(stid);
      RETURN(stid);
      END.

   (getvnd) %find end of branch%                                   9A8
      % This procedure finds the "end" of the branch begun by
      the stid passed it; it expects, as a second argument, a
      maximum level to be used in determining the end of the
      branch.  If all statements in the branch fall beneath the
      minimal level, it returns the the stid passed it. %
      %-------------------%
      PROC(stid, level);
      LOCAL curlev, tmstid;
      curlev _ getlev(stid);
      UNTIL curlev >= level DO
         BEGIN
         IF (stid := getsub(stid)) = stid THEN RETURN(stid);
         stid _ getail(stid);
         BUMP curlev;
         END;
      RETURN(stid);
      END.

   (getbck)                                                      9A9
      %This procedure finds the back of the stid of the statemment
      passed it.  It does not observe viewspecs.%
      %-------------%
      PROCEDURE(stid);
      IF stid.stpsid = origin OR getorf(stid) THEN RETURN(stid);
      IF getfhd(stid) THEN RETURN(getup(stid));
      stid _ getprd(stid);
      IF (stid := getsub(stid)) = stid THEN RETURN(stid);
      RETURN(getend(getail(stid)));
      END.

   (getnxt)                                                      9A10
      %This procedure finds the sequentially "next" statement, i.e.
      the substatement, successor, or successor of up, etc, of the
      stid passed as argument.  Ignores all viewspecs.%
      %---------------%
      PROCEDURE (stid);
      IF stid = endfil THEN err($"end of file exceeded");
      IF (stid := getsub(stid)) = stid THEN
         % no substructure %
         BEGIN
         LOOP
            BEGIN
            IF stid.stpsid = origin OR getorf(stid) THEN RETURN
            (endfil);
            IF getftl(stid) = 0 THEN EXIT; % not a tail %
            stid _ getsuc(stid);
            END;
         stid _ getsuc(stid);
         END;
      IF stid.stpsid = origin OR getorf(stid) THEN RETURN (endfil);
      RETURN (stid);
      END.


% Get --  flag values and field contents in ring %
   (getftl) %The logical value of the tail flag is returned.%
                                                                 9B1
      PROCEDURE (stid);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      RETURN ([rngloc].rtf) END.

   (getfhd) %The logical value of the head flag is returned.%
                                                                 9B2
      PROCEDURE (stid);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      RETURN ([rngloc].rhf) END.
   (getnmf) %The logical value of the name flag is returned.%
                                                                      9B3
      PROCEDURE (stid);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      RETURN ([rngloc].rnamef) END.

   (getnam) %The hash word for the statement name is returned.%
                                                                      9B4
      PROCEDURE (stid);
      LOCAL rngloc;
      lodent( stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      RETURN ([rngloc].rnameh) END.

   (getsid) %The statement identifier for the statement name is
   returned.%
                                                                      9B5
      PROCEDURE (stid);
      LOCAL rngloc;
      lodent( stid, rngtyp : rngloc);
      %this does NOT call err if sid=0 so that sid can be checked by
      other programs to test a ring element for validity%
      RETURN ([rngloc].rsid) END.

% Get first property stdb %
   (getsdb) %The STDB associated with  the specified ring element is
   returned.  NOT TO BE USED TO GET THE STDB TO WHICH AN INFERIOR
   TREE IS CONNECTED!  USE GETPHED INSTEAD.  If the node is the
   origin of an inferior tree, this procedure returns FALSE.%
                                                                      9C1
      PROCEDURE (stid);
      LOCAL stdb, rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      IF [rngloc].rtorgin THEN RETURN(0);
      stdb _ 0;
      IF (stdb.stpsdb _ [rngloc].rsdb) = 0 THEN RETURN(0);
      stdb.stfile _ stid.stfile;
      RETURN (stdb) END.

% Get next property stdb %
   (getpsdb) PROCEDURE %***% ( stdb );                                9D1
      % Given an stdb, this procedure finds the spsdb field of the
      property (the next in the chain and converts it into an stdb;
      if it is zero it returns zero. %
      LOCAL nxtprop, dbloc;
      lodent( stdb, sdbtyp : dbloc);
      nxtprop _ 0;
      IF (nxtprop.stpsdb _ [dbloc].spsdb) = 0 THEN RETURN(0);
      nxtprop.stfile _ stdb.stfile;
      RETURN (nxtprop);
      END.

% Get next property stdb %
```

```
   (getitree) PROCEDURE %***% ( stdb );                                9E1
      % Given an stdb, this procedure finds the sitpsid field of the
      property (the inferior tree) and converts it into an stid; if
      it is zero it returns zero. %
      LOCAL itstid, dbloc;
      lodent( stdb, sdbtyp : dbloc);
      itstid _ 0;
      IF (itstid.stpsdb _ [dbloc].sitpsid) = 0 THEN RETURN(0);
      itstid.stfile _ stdb.stfile;
      RETURN (itstid);
      END.

% Get -- field values from data blocks %

   (getorg) %***%                                                      9F1
      %Given an stid, this routine returns the origin of the node%
      %-------------%
      PROCEDURE(stid);
      LOCAL rnl;
      REF rnl;
      LOOP
         BEGIN
         IF stid.stpsid = origin OR getorf( stid ) THEN
            EXIT LOOP;
         stid _ getup(stid);
         END;
      RETURN(stid);
      END.

   (getorf) %***% %The logical value of the origin flag is
   returned.%
                                                                       9F2
      PROCEDURE (stid);
      LOCAL rngloc;
      lodent( stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      RETURN ([rngloc].rtorgin) END.

   (getnmdl) %***% PROCEDURE (stid);                                   9F3
      LOCAL
         dlleft,
         dlright,
         sdb;
      REF sdb;
      IF NOT lodprop( stid, txttyp: &sdb) THEN
         BEGIN
         % No text block associated: no name delimiters.  Error? %
         dlieft _ dlright _ 0;
         END
      ELSE
         BEGIN
         dlleft _ sdb.slnmdl;
         dlright _ sdb.srnmdl;
         END;
      RETURN(dlleft, dlright);
      END.
```

```
(getphed) %***%                                                    9F4
   %Given an stid  (hopefully in an inferior tree), this routine
   returns the stdb of the property to which is attached.  If not
   in an inferior tree, returns false%
   %-------------%
   PROCEDURE(stid);
   LOCAL stdb, rngloc;
   IF stid.stpsid # origin THEN
      BEGIN
      stid _ getorg(stid);
      IF stid.stpsid = origin THEN RETURN(FALSE);
      % The psdb field of the origin of an inferior tree points
      to the  property header. %
         lodent(stid, rngtyp : rngloc);
         IF [rngloc].rsid = 0 THEN err($"Bad statement
         identifier");
         IF [rngloc].rtorgin = 0 THEN err($"Inferior tree origin
         error");
         stdb _ 0;
         IF (stdb.stpsdb _ [rngloc].rsdb) = 0 THEN RETURN(0);
         stdb.stfile _ stid.stfile;
         RETURN(stdb);
      END
   ELSE RETURN(FALSE);
   END.

(getint) %***% %Called with STDB, returns initials of associated
statement (left justified in word zero filled).%
                                                                   9F5
   PROCEDURE (stdb);
   LOCAL STRING string[5];
   LOCAL sdbint, sdbloc;
   lodent( stdb, sdbtyp : sdbloc);
   *string* _ NULL;
   string[1] _ 0;
   IF (sdbint _ [sdbloc].sinit) .A 4B6 THEN %old syle inits%
      string[1].oldint _ sdbint
   ELSE trnsint(sdbint, $string);
      %sdb initials in five-bit characters%
   RETURN(string[1]);
   END.

(trnsint) PROCEDURE(initials, string);                             9F6
   %Given a word of 5-bit initials, this routine translates them
   and appends them to the string whose address is passed.%
   REF string;
   LOCAL char, charno, count;
   count _ 0;
   UNTIL (count _ count + 1) > 4 DO
      BEGIN
      charno _ CASE count OF
         =1: initials.cint1;
         =2: initials.cint2;
         =3: initials.cint3;
         ENDCASE initials.cint4;
      IF (char _ *trn5t7*[charno + 1]) = SP THEN EXIT LOOP
```

```
         ELSE *string* _ *string*, char;
         END;
      RETURN;
      END.

   (gettim) %***% %Called with stdb, returns time of last write for
   associated statement.%
                                                                   9F7
      PROCEDURE (stdb);
      LOCAL sdbloc;
      lodent( stdb, sdbtyp : sdbloc);
      RETURN ([sdbloc].stime) END.

   (getdblen) %***% %Called with stdb, returns length of data block%
                                                                   9F8
      PROCEDURE (stdb);
      LOCAL sdbloc;
      lodent( stdb, sdbtyp : sdbloc);
      RETURN ([sdbloc].slength) END.

   (getstsize) %***% %Called with STID, returns size in characters
   for associated statement.%
                                                                   9F9
      PROCEDURE (stid);
      LOCAL sdbloc;
      IF NOT lodprop( stid, txttyp : sdbloc) THEN
         err($"No text block associated with this node; file
         probably bad.");
      RETURN ([sdbloc].schars) END.

   (fchtxt)%***% %Called with stid returns the number of the first
   character after the name of the corresponding text block .  MUST
   CHANGE CALLS ON THIS.  IF NO TEXT BLOCK, generates an error.%
                                                                   9F10
      PROCEDURE (stid);
      LOCAL sdbloc;
      IF NOT lodprop (stid, txttyp : sdbloc) THEN
         err($"No text block associated with this statement; file
         probably bad");
      RETURN ([sdbloc].sname) END.

% Store into ring and data block fields %

   (stosuc) %Store successor of specified statement.%
                                                                   9G1
      PROCEDURE (stid, sucstid);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      [rngloc].rsuc _ sucstid.stpsid;
      RETURN END.

   (stosub) %Store sub of specified statement.%
                                                                   9G2
      PROCEDURE (stid, substid);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      [rngloc].rsub _ substid.stpsid;
      RETURN END.

   (stoftl) %Store tail flag of specified statement.%
                                                                  9G3
      PROCEDURE (stid, tail);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      [rngloc].rtf _ tail;
      RETURN END.

   (stofhd) %Store head flag of specified statement.%
                                                                  9G4
      PROCEDURE (stid, head);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      [rngloc].rhf _ head;
      RETURN END.

   (stonmf) %Store name flag of specified statement.%
                                                                  9G5
      PROCEDURE (stid, name);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      [rngloc].rnamef _ name;
      RETURN END.

   (stosdb) %Store the psdb of specified statement.%
                                                                  9G6
      PROCEDURE (stid, stdb);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      [rngloc].rsdb _ stdb.stpsdb;
      RETURN END.

   (stonam) %Store the name hash of specified statement.%
                                                                  9G7
      PROCEDURE (stid, nameh);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
      [rngloc].rnameh _ nameh;
      RETURN END.

   (stosid) %Store the sid of specified statement.%
                                                                  9G8
      PROCEDURE (stid, sid);
      LOCAL rngloc;
      lodent(stid, rngtyp : rngloc);
      IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
```

```
         [rngloc].rsid _ sid;
         RETURN END.

      (stoorg) %***% %Store origin flag of specified statement.%
                                                                   9G9
         PROCEDURE (stid, orgflg);
         LOCAL rngloc;
         lodent(stid, rngtyp : rngloc);
         IF [rngloc].rsid = 0 THEN err($"Bad statement identifier");
         [rngloc].rtorgin _ orgflg;
         RETURN END.

      (stoitree) %***% %Store psid of inferior tree in the data block
      whose stdb is passed.%
                                                                   9G10
         PROCEDURE (stdb, psid);
         LOCAL dbloc;
         lodent(stdb, sdbtyp : dbloc);
         [dbloc].sitpsid _ psid;
         RETURN END.

 % basic entity structural edits  -- core level %
   % Load entity %

      (lodent) %***% %This routine loads file block (structural or
      data) into core given its STID or STDB and the type of block,
      RNGTYP or SDBTYP.  Returns                                   10A1
         1) the CRPGAD index for the new page
         2) the address of the sdb  or ring in core.%

         PROCEDURE (flbadr, blktyp);
         LOCAL
            wc,      %word count in block to RNL or  SDB%
            pgindx, %page index where loaded%
            tbladr,  %start of appropriate status table. %
            blkbase,   % base of block %
            st;      %pointer to RNGST or DTBST entry for the page%
         REF st;
         CASE blktyp OF
            = rngtyp:
               BEGIN
               tbladr _ $rngst;
               blkbase _ rngbas;
               END;
            = sdbtyp:
               BEGIN
               tbladr _ $dtbst;
               blkbase _ dtbbas;
               END;
            ENDCASE err($"Illegal blktyp passed to lodent");
         % this needs to be fast -- so we'll cheat and get the fields
         out of the flbadr without using byte pointers.  the record
         definitions for stid and stdb in utilty have a comment to the
         effect that we are doing this. %
         R2 _ flbadr;
         !HLRZ R1,R2;
      %the stfile field is right justified in the left halfword%
      !HRRZ R2,R2;  %clear the left half%
      !IDIVI R2,1B3; %split out the block number and the word count%
      wc _ R3;  % the rngblk or dtbblk number is in R2 %
      &st _ filehead[R1] + tbladr - $filhed + R2;
      %load if necessary%
      IF (pgindx _ st.rfcore) = 0 THEN %must load it%
         pgindx _ lodrfb(R2+blkbase, blktyp, R1);
      RETURN (pgindx, crpgad[pgindx]+wc);
      END.

   (lodprop) % xxx  load property %
   PROCEDURE (stid, proptyp);                                      10A2
      % loads the indicated property block into core.  Returns three
      items: first is FALSE if error, page number in core if
      success; second is address of block in core (which must be
      frozen if you want to do anything with it!), third is stdb of
      property block %
      %--------------%
      LOCAL
         ptabin,
         stdb,
         blknum,
         db;
      REF db;
      IF stid.stastr THEN err($"String treated as statement");
      % getsdb gets the first data block associated with a ring %
      IF NOT (stdb _ getsdb(stid)) THEN
         RETURN( FALSE, 0, 0);
      ptabin _ getptab(proptyp);
      LOOP
         BEGIN
         blknum _ lodent(stdb, sdbtyp : &db);
         IF db.sptype = proptyp THEN EXIT LOOP;
         IF  (getptab(db.sptype) > ptabin) OR (stdb.stpsdb _
         db.spsdb) = 0 THEN
            RETURN(FALSE, 0, 0);
         END;
      RETURN(blknum, &db, stdb);
      END.

   (lodrfb) %Load random file block.  Arguments are                10A3
      1) file block number,
      2) block type,
         or negative number if just want to get a core page,
      3) file number.
      Returns page index (i.e. the index in CRPGAD and CORPST
      for the page).  Look in CRPGAD indexed to get the page
      address.  This is NOT called to load the header of a
      file -- rdhdr does that job.%
      PROCEDURE (nblk, btype, fileno);
      LOCAL
         pgindex, %file block page index%
         blk,     %block address of the page%
         fl,      %file list header%
         jfn,     %file number for jsyses%
      access,  %read/write access to the page%
      stent,   %pointer to status table entry%
      ct;      %used as pointer into CORPST%
   REF ct, blk, fl, stent;
   IF nxpg NOT IN [rfpmin,rfpmax] THEN nxpg _ rfpmin;
   %choose the next file block page to use%
   &ct _ $corpst + pgindex _ nxpg;
   % look for an empty page %
   WHILE ct.ctfull DO
      BEGIN
      IF (pgindex _ pgindex+1) > rfpmax THEN
         &ct _ $corpst + pgindex _ rfpmin
      ELSE BUMP &ct;
      IF pgindex = nxpg THEN
         BEGIN %have checked all the pages. none empty%
         %search for an unfrozen page%
         WHILE ct.ctfroz DO BEGIN
            IF (pgindex _ pgindex+1) > rfpmax THEN
               &ct _ $corpst + pgindex _ rfpmin
            ELSE BUMP &ct;
            IF pgindex = nxpg THEN rerror(); %all frozen%
            END;
         IF ct.ctfile > 0 THEN %does belong to some file%
            BEGIN
            %revise the appropriate status table%
            &stent _ filehead[ct.ctfile] + $rfbs - $filhed
            + ct.ctpnum;
            stent.rfcore _ 0;
            END;
         ct.ctfull _ FALSE;
         END;
      END;
   IF (nxpg _ pgindex+1) > rfpmax THEN nxpg _ rfpmin;
   %set up corpst for the new page%
   ct _ 0;
   ct.ctfull _ TRUE;
   IF btype < 0 THEN %just getting a page%
      BEGIN
      %remove the page that is there now%
      R2.LH _ 4B5;
      R2.RH _ crpgad[pgindex] / 512;
      !pmap( -1, R2, 0);
      RETURN (pgindex);
      END;
   ct.ctfile _ fileno;
   ct.ctpnum _ nblk;
   &stent _ filehead[fileno] + $rfbs - $filhed + nblk;
   &fl _ (fileno - 1)*filstl + $filst;
   IF btype = niltyp AND fl.flpart OR stent.rfpart THEN
      BEGIN
      jfn _ fl.flpart;
      access _ 14B10; %read-write%
      END
   ELSE
      BEGIN
      jfn _ fl.florig;
```

```
         access _ 1B11; %read only%
         END;
      IF jfn = 0 THEN
         BEGIN
         ct _ 0;
         IF NOT fl.flexis THEN err($"fst entry nonexistant");
         err($"Illegal JFN in LODRFB");
         END;
      %set up registers for PMAP%
      &blk _ crpgad[pgindex];
      R1.LH _ jfn;
      R1.RH _ nblk;
      R2.LH _ 4B5;
      R2.RH _ &blk / 512;
      !pmap(R1, R2, access);
      stent.rfcore _ pgindex;
      IF btype = niltyp THEN %do the initialization%
         BEGIN
         IF fl.flpart THEN stent.rfpart _ TRUE;
         blk.fbpnum _ nblk;
            %rest is done by the calling procedure%
         END
      ELSE %verify%
         IF blk.fbtype # btype OR blk.fbpnum # nblk THEN
            badfil(fileno);
      RETURN (pgindex) END.

* nodes and groups of nodes %
   (crenod) % xxx  create node%
   PROCEDURE ( stid, rlevcnt );                                10B1
      % Gets a new ring element with no associated data blocks and
      links it into the structure at the location specified by stid
      and rlevcnt (a relative level count: < 0 is down, =0 is
      successor, > 0 is up by rlevcnt levels).  Returns stid of new
      ring or 0 if error. %
      %---------------%
      LOCAL newstid;
      INVOKE (sigrtn, rtnf);
      newstid _ newrng( stid.stfile );
      insgrp( stid, rlevcnt, newstid, newstid);
      DROP (sigrtn);
      RETURN(newstid);
      (rtnf):                                                  10B1I
      DROP (sigrtn);
      RETURN (FALSE);
      END.

   (copgrp) %****%                                             10B2
      %Given an stid as the first argument, this routine copies the
      group bounded by the second and third arguments after the
      first stid, in the direction specified by the fourth argument.
       The fifth argument indicates whether the correspondence list
      should be updated.
         It proceeds by constructing new ring elements for each
         branch defined by the top-level statements in the group.
         Then it copies these data blocks into freshly allocated
         SDB's.  It then inserts the newly created group after the
         stid passed it.%
      %-------------%
      PROCEDURE(stid, dir, grp1, grp2, upctf);
      LOCAL newsid, %new stid%
         oldsid, %old stid being processed%
         newgp1; %head of new group%
      IF grp1.stfile # grp2.stfile THEN err($"illegal group");
      oldsid _ grp1;
      newsid _ newgp1 _ newrng(stid.stfile);
      LOOP
         BEGIN
         %get stid's, this branch%
         WHILE (oldsid := getsub(oldsid)) # oldsid DO
            newsid _ newsub(newsid);
         %now copy property lists and point to next branch%
         LOOP
            BEGIN
            copplist(oldsid, newsid);
            IF upctf THEN upctbl(oldsid, newsid, oldsid);
            IF oldsid = grp1 THEN %branch copy complete%
               BEGIN
               IF oldsid = grp2 THEN %group copy complete%
                  BEGIN
                  insgrp(stid, dir, newgp1, newsid);
                  RETURN(newgp1, newsid);
                  END;
               oldsid _ grp1 _ getsuc(oldsid);
               EXIT;
               END;
            IF NOT getftl(oldsid := getsuc(oldsid)) THEN EXIT;
               %still more in plex%
            newsid _ getsuc(newsid);
            END;
         newsid _ newsuc(newsid);
         END;
      END.

   (copfgrp) PROCEDURE        % copy filtered group %                  10B3
      (target, rlevcnt, src1, src2, vsptr);
      LOCAL retstid, newtgt, newsrc, lstlev, toplev, sqgwrk, vspec1,
      vspec2, usqcod, cacode;
      LOCAL TEXT POINTER tp1, tp2;
      REF sqgwrk, vsptr;

      * initialize %
         vspec1 _ vsptr.vs1;
         vspec2 _ vsptr.vs2;
         cacode _ vsptr.vscacode;
         usqcod _ vsptr. vsusqcod;
         retstid _ target;
      &sqgwrk _ openseq (src1, src2, vspec1, vspec2, usqcod,
      cacode);
      INVOKE (sigcls);
      IF (newsrc _ seqgen(&sqgwrk)) # endfil THEN
         BEGIN
         IF newsrc.stastr THEN
            BEGIN
            FIND SF(newsrc) ^tp1 SE(newsrc) ^tp2;
            retstid _ newtgt _ cinssta( target, rlevcnt, $tp1, $tp2
            )
            END
         ELSE retstid _ newtgt _
            ccopsta( target, rlevcnt, newsrc, FALSE, FALSE);
         toplev _ lstlev _
            IF sqgwrk.swclvl = 0 THEN 1 ELSE sqgwrk.swclvl;
         WHILE (newsrc _ seqgen(&sqgwrk)) # endfil DO
            BEGIN
            rlevcnt _ 0;
            CASE sqgwrk.swclvl OF
               =lstlev: NULL;
               >lstlev:
                  BEGIN
                  rlevcnt _ -1;
                  lstlev _ lstlev + 1;
                  END;
               <lstlev:
                  WHILE lstlev > toplev DO
                     BEGIN
                     BUMP rlevcnt;
                     IF ((lstlev _ lstlev - 1) = sqgwrk.swclvl) THEN
                        EXIT LOOP 1;
                     END;
               ENDCASE NULL;
            IF newsrc.stastr THEN
               BEGIN
               FIND SF(newsrc) ^tp1 SE(newsrc) ^tp2;
               newtgt _ cinssta( newtgt, rlevcnt, $tp1, $tp2 )
               END
            ELSE newtgt _
               ccopsta( newtgt, rlevcnt, newsrc, FALSE, FALSE);
            END;
         END;
      closeseq (&sqgwrk := 0);
      DROP (sigcls);
      RETURN(retstid, newtgt);

      (sigcls) CATCHPHRASE;                                       10B3N
          IF &sqgwrk THEN
            BEGIN
            DISABLE (sigcls);
            IF &sqgwrk THEN closeseq( &sqgwrk := 0 );
            CONTINUE;
            END;

      END.

   (mvdlfgrp)         % move / delete filtered group %            10B4
      PROCEDURE
         (stid1,      %stid of first statement to be moved/deleted%
         stid2,       %stid of second statement to be moved deleted%
         vsptr,       %pointer to filter viewspec record%
         newstid,     %stid of statement following group: stid1
         stid2%
         type,        %delete/move/transpose%
         mvdlflag,    %stid of target for move/transpose%
         mvrlev       %relative level for move/transpose%
         );
      LOCAL
         vspec1, unfseq, ufstid, ufring, filseq, filstid, filring,
         cring, pflag, rstid, toplev, lstlev, mtoplev, mlstlev,
         rlevcnt, cstid, nstid;
      REF vsptr, unfseq, ufring, filseq, filring, cring;

      vspec1 _ vsptr.vs1;
         vspec1.vslev _ -1;  % all levels %
         vspec1.vscapf _ FALSE;  % no content analyzer program %
         vspec1.vsusqf _ FALSE;  % no sequence generator program %
      &unfseq _ &filseq _ 0;
      % make unfiltered pass thru group; mark all with absolute
      level %
         % open an unfiltered sequence %
            &unfseq _
               openseq( stid1, stid2, vspec1, vsptr.vs2, FALSE,
               FALSE);
         INVOKE (sigcls);
         WHILE (ufstid _ seggen(&unfseq)) # endfil DO
            BEGIN
            lodent( ufstid, rngtyp : &ufring);
            ufring.rinst1 _ unfseq.swclvl;
            ufring.rdummy _ TRUE;
            END;
         closeseq( &unfseq := 0 );
      % make filtered pass marking all statements that pass false %
         % open a filtered sequence %
            &filseq _
               openseq( stid1, stid2, vsptr.vs1, vsptr.vs2,
               vsptr.vsusqcod, vsptr.vscacode);
         WHILE (filstid _ seggen(&filseq)) # endfil DO
            BEGIN
            lodent(filstid, rngtyp : &filring);
            filring.rdummy _ FALSE;
            END;
         closeseq( &filseq := 0 );
         DROP (sigcls);
      % get predessor or up (remember which) of first unfiltered %
         lodent( stid1, rngtyp : &cring);
         pflag _ IF cring.rhf THEN FALSE ELSE TRUE;
         rstid _ IF pflag THEN getprd(stid1) ELSE getup(stid1);
      % remove the entire group %
         IF NOT remgrp( stid1, stid2) THEN
            CASE type OF
               = moveflag: err($"illegal move");
               = trnsflag: err($"illegal transpose");
               = deltflag: err($"illegal delete");
               ENDCASE;
      % initialize level stuff %
         toplev _ lstlev _ mtoplev _ mlstlev _ 0;
```

```
        % initialize for main loop %
           cstid _ stid1;
        % final pass through removed group %
           WHILE cstid DO
              BEGIN
              % get ring element for current stid %
                 lodent( cstid, rngtyp : &cring);
              % set up stack for next statement %
                 IF cstid = stid2 THEN
                    BEGIN
                    nstid _ 0;
                    !PUSH S,nstid;
                    END
                 ELSE IF NOT cring.rtf THEN
                    BEGIN
                    nstid _ cring.rsuc;
                    nstid.stfile _ stid1.stfile;
                    !PUSH S,nstid;
                    END;
                 cring.rsuc _ 0;
                 IF (nstid_ cring.rsub:= cstid.stpsid) # cstid.stpsid
                 THEN
                    BEGIN
                    nstid.stfile _ stid1.stfile;
                    !PUSH S,nstid;
                    END;
              IF cring.rdummy THEN
                 CASE type OF
                    = trnsflag: delgrp(cstid, cstid, newstid);
                    ENDCASE
                       BEGIN   % insert the statement %
                       IF NOT toplev THEN
                          BEGIN
                          toplev _ lstlev _ cring.rinst1;
                          rlevcnt _ IF pflag THEN 0 ELSE -1;
                          END
                       ELSE rlevcnt _ 0;
                       CASE cring.rinst1 OF
                          = lstlev:  NULL;
                          > lstlev:
                             BEGIN
                             rlevcnt _ -1;
                             BUMP lstlev;
                             END;
                          < lstlev:
                             WHILE lstlev > toplev DO
                                BEGIN
                                BUMP rlevcnt;
                                IF ((lstlev _ lstlev - 1) =
                                cring.rinst1) THEN
                                   EXIT LOOP 1;
                                END;
                          ENDCASE;
                       insgrp( rstid := cstid, rlevcnt, cstid, cstid
                       );
                       END
```

```
         ELSE
            CASE type OF
               = deltflag: delgrp(cstid, cstid, newstid);
               ENDCASE
                  BEGIN   % move the statement %
                  IF NOT mtoplev THEN
                     BEGIN
                     mtoplev _ mlstlev _ cring.rinst1;
                     rlevcnt _ mvrlev;
                     END
                  ELSE rlevcnt _ 0;
                  CASE cring.rinst1 OF
                     = mlstlev:  NULL;
                     > mlstlev:
                        BEGIN
                        rlevcnt _ -1;
                        BUMP mlstlev;
                        END;
                     < mlstlev:
                        WHILE mlstlev > mtoplev DO
                           BEGIN
                           BUMP rlevcnt;
                           IF ((mlstlev _ mlstlev - 1) =
                           cring.rinst1) THEN
                              EXIT LOOP 1;
                           END;
                     ENDCASE;
                  insgrp( mvdlflag := cstid, rlevcnt, cstid,
                  cstid );
                  END;
         % get next stid %
            !POP S,cstid;
         END;
   RETURN;

   (sigcls) CATCHPHRASE;                                        10B4P
      BEGIN
      DISABLE (sigcls);
      IF &unfseq THEN closeseq (&unfseq := 0);
      IF &filseq THEN closeseq (&filseq := 0);
      CONTINUE;
      END;
   END.
(remgrp)                                                        10B5
   %Removes group whose bounds are passed it from position in
   ring but does not delete either ring elements or SDB's.%
   %-------------%
   PROCEDURE(grp1, grp2);
   LOCAL stid; %work area for updating stid%
   IF grp1.stpsid = origin OR
      grp2.stpsid = origin THEN RETURN(FALSE);
   IF grp1.stfile # grp2.stfile THEN err($"illegal group");
   %putcdpanic(IF getfhd(grp1) THEN getup(grp1)
      ELSE getprd(grp1));%
   IF getfhd(grp1) THEN
      BEGIN
```

```
          stid _ getsuc(grp2); %new plex head%
          IF NOT getftl(grp2) THEN stofhd(stid, TRUE);
          stosub(getup(grp2), stid);
          END
       ELSE
          BEGIN
          stid _ getprd(grp1);
          stoftl(stid, getftl(grp2));
          stosuc(stid, getsuc(grp2));
          END;
       RETURN;
       END.

    (delgrp) %****%                                                     10B6
       %This routine destroys all trace of the group bounded by the
       arguments passed it.  (Note that it does not remove the group
       from the ring; it assumes that this has been done by, for
       example, REMGRP.)
          It follows each branch to its deepest level, deleting
          references to substructure in the sub fields; it then
          follows the structure back up, through the suc pointers,
          freeing SDB's as it goes.%
       %-------------%
       PROCEDURE(grp1, grp2, newsid);
       LOCAL
          stid, %stid being processed%
          subsid; %sub of stid being processed%
       IF grp1.stfile # grp2.stfile THEN err($"illegal group");
       IF grp1.stpsid = origin OR
          grp2.stpsid = origin THEN err($"illegal delete");
       stid _ grp1;
       LOOP
          BEGIN
          WHILE (subsid _ getsub(stid)) # stid DO
             BEGIN
             stosub(stid, stid);
             stid _ subsid;
             END;
          stid _ getsuc(subsid); %now go back up%
          upctbl(subsid, endfil, newsid);
          freplist(subsid); %free property list%
          frerng(subsid);
          IF subsid = grp2 THEN RETURN;
          END;
       END.

% property list %
    (copplist) % xxx ***% %Copies a property list.  Arguments are 10C1
       1) source STID,
       2) destination STID.
       It is assumed that destination does NOT have an SDB currently
       associated with it.%

       PROCEDURE (source, dest);
       LOCAL
          db, % address in core of current data block %
```

```
         oldb; % stdb of current data block %
      REF db;
      % find stdb of first property block to be copied.  returns 0
      if origin of inferior tree. %
         IF NOT (oldb _ getsdb(source)) THEN RETURN;
      LOOP
         BEGIN
         % load the current property block into core to get its type
         %
            lodent( oldb, sdbtyp : &db );
         % Copy the property; SIGNAL if error occurs %
            IF NOT (copprop( source, db.sptype, dest)) THEN
               err($"File system error on copy.  File may be bad.");
         % get the next property  stdb: returns false if no more in
         list %
            IF NOT (oldb _ getpsdb(oldb)) THEN EXIT LOOP;
         END;
      RETURN END.

   (freplist) %***% PROCEDURE (stid);                            10C2
      % free all properties associated with this node.  calls
      treprop until the last property is reached %
      LOCAL
         rnl,
         sdb,
         next,
         fileno,
         sdbit; % stid of inferior tree of property freed; it must
         be freed as well %
      REF rnl, sdb;
      lodent(stid, rngtyp : &rnl);
      IF rnl.rtorgin THEN RETURN;
      IF (next _ rnl.rsdb) = 0 THEN RETURN;
      fileno _ stid.stfile;
      LOOP
         BEGIN
         next.stfile _ fileno;
         lodent(next, sdbtyp : &sdb);
         IF sdb.sptype = txttyp THEN
            BEGIN
            % reset this string in correspondence table before
            getting rid of it %
            FIND SF(stid) ^sptr1 SE(stid) ^sptr2;
            cldsr ($sptr1);
            END;
         IF (sdbit _ freprop( next := sdb.spsdb)) THEN
            freintree( sdbit );
         IF NOT next THEN EXIT LOOP;
         END;
      stosdb( stid, 0);
      RETURN;
      END.

% property %
   (creprop) % xxx  Create property block %
   PROCEDURE ( stid, proptyp, length, data );                    10D1
```

```
      % Builds a data block of property type proptyp (which must be
      a valid type atssigned (and declared ) by ARC and links it
      into the plist associated with the stid in the proper order
      (determined in the procedure locprop).  If such a property
      already exists in the node, we have an error:  it must first
      be deleted.  Returns stdb of new block or 0 if error.  length
      is the length of the data and data is a pointer to an array of
      length words in which the data is stored.  If &data is zero,
      do not copy data; just initialize the block. Calls crepr2 to
      do actual work. %
      %---------------%
      RETURN( crepr2( stid, proptyp, length, data, FALSE, 0, 0) );
      END.

   (copprop) %xxx copy property%
   PROCEDURE ( stid, % of source %                                    10D2
      proptyp,
      destid % of destination node % );
      % copies property block (and associated inferior tree if any)
      from block indicated by stid and proptyp to a new block to be
      created on destid.  Returns TRUE if OK. 0 if error %
      %---------------%
      LOCAL
         sdbblk, % page number with source sdb %
         sdbold, % address of source sdb %
         stdb, % stdb of source sdb %
         destblk, % page number with destination (ring or sdb) %
         desel, % address of destination ring or sdb %
         dest, %stdb or stpsid of destination %
         destrng, % TRUE if destination a ring, FALSE if a property
         %
         nwblk, % page with new db %
         sdbnew, % address of new db %
         nwtdb, % stdb of new sdb %
         sdbfrz, destfrz, nwfrz,
         room, % size of source db %
         intre; % stid of origin of inferior tree %
      REF sdbold, desel, sdbnew;
      sdbfrz _ destfrz _ nwfrz _ 0;
      % Get and freeze source block %
         INVOKE (sigrtn, rtnf2);
         IF NOT (sdbblk _ lodprop(stid, proptyp : &sdbold, stdb ))
         THEN
            BEGIN
            DROP (sigrtn);
            RETURN( FALSE );
            END;
         frzblk( sdbblk, 1);
         sdbfrz _ TRUE;
      % Get and freeze destination; locprop finds the proper
      location for this new block. %
         IF NOT (destblk _ locprop(destid, proptyp : &desel, dest,
         destrng)) THEN
            BEGIN
            DROP (sigrtn);
            IF (sdbfrz := FALSE) THEN frzblk(sdbblk, -1);
            RETURN(FALSE);
            END;
         frzblk( destblk, 1);
         destfrz _ TRUE;
      % Get a new block %
         room _ sdbold.slength;
         nwtdb _ newdb(room, proptyp, dest.stfile : &sdbnew, nwblk);
         frzblk( nwblk, 1);
         nwfrz _ TRUE;
      % copy data from source to new block %
         mvbfbf( &sdbold, &sdbnew, room);
      % copy name fields from old block to new %
         copnam( destrng, &sdbold, stid, &desel, dest);
      % link in the property. lnkprop assumes appropriate blocks are
      in core and frozen.  locprop has been used to find the correct
      location for the new property block %
         lnkprop( destrng, &desel, dest, &sdbnew, nwtdb);
      % Unfreeze destination and source %
         sdbfrz _ 0;
         frzblk(sdbblk, -1);
         destfrz _ 0;
         frzblk(destblk, -1);
         nwfrz _ 0;
      % copy inferior tree if any.  Put the copy's STPSID in the
      SITPSID field of the new block. %
         IF sdbnew.sitpsid THEN
            BEGIN
            intre _ sdbnew.sitpsid;
            intre.stfile _ stid.stfile;
            IF NOT trecop( intre, nwtdb) THEN
               BEGIN
               DROP (sigrtn);
               RETURN( FALSE );
               END;
            % trecop also links the tree to nwtdb %
            END;
      frzblk(nwblk, -1);
      DROP (sigrtn);
      RETURN(nwtdb);
      (rtnf2):                                                   10D2S
      DROP (sigrtn);
      IF (sdbfrz:=0) THEN frzblk(sdbblk, -1);
      IF (destfrz:=0) THEN frzblk(destblk, -1);
      IF (nwfrz:=0) THEN frzblk(nwblk, -1);
      RETURN (FALSE);
      END.

   (movprop) % xxx  move property %
   PROCEDURE ( stid, % of source %                               10D3
      proptyp,
      destid % of destination node % );
      % Moves the property indicated from node specifed by stid to
      node specified by destid.  Accomplishes this by unlinking and
      relinking the block.  If a property type of the type being
      moved exists at the destination, we have an error.  Returns
      true if OK, 0 if error. %
```

```
%---------------%
LOCAL
   sdbblk, % page number with source sdb %
   sdbold, % address of source sdb %
   stdb, % stdb of source sdb %
   destblk, % page number with destination (ring or sdb) %
   desel, % address of destination ring or sdb %
   dest, %stdb or stpsid of destination %
   sdbfrz, destfrz,
   destrng; % TRUE if destination a ring, FALSE if a property
   %
REF sdbold, desel;
sdbfrz _ destfrz _ 0;
% If two files involved, copy property to new node. %
   IF stid.stfile # destid.stfile THEN
      BEGIN
      % ---- INTERFILE MOVES ---- %
      % Copy block to new file; copprop links it in %
         IF NOT (copprop(stid, proptyp, destid )) THEN
            RETURN( FALSE );
      % delete the original %
         IF NOT( delprop( stid, proptyp )) THEN
            RETURN( FALSE );
      RETURN( TRUE );
      END;
% ---- INTRAFILE MOVES ---- %
% Get and freeze source block %
   INVOKE (sigrtn, rtnf3);
   IF NOT (sdbblk _ lodprop(stid, proptyp : &sdbold, stdb ))
   THEN
      BEGIN
      DROP (sigrtn);
      RETURN( FALSE );
      END;
   frzblk( sdbblk, 1);
   sdbfrz _ TRUE;
% Get and freeze destination; locprop finds the proper
location for this new block. %
   IF NOT (destblk _ locprop(destid, proptyp : &desel, dest,
   destrng)) THEN
      BEGIN
      IF (sdbfrz:=0) THEN frzblk(sdbblk, -1);
      DROP (sigrtn);
      RETURN(FALSE);
      END;
   frzblk( destblk, 1);
   destfrz _ TRUE;
% copy name fields from old to new ring %
   copnam( destrng, &sdbold, stid, &desel, dest);
% Unlink current property location; relink nodes around it. %
   IF NOT (unlnkprop( &sdbold,  stdb)) THEN
      BEGIN
      IF (sdbfrz:=0) THEN frzblk(sdbblk, -1);
      IF (destfrz:=0) THEN frzblk(destblk, -1);
      DROP (sigrtn);
      RETURN(FALSE);
```

```
          END;
      % link in the property. lnkprop assumes appropriate blocks are
      in core and frozen.  locprop has been used to find the correct
      location for the new property block %
         lnkprop( destrng, &desel, dest, &sdbold, stdb);
      % Unfreeze destination and source %
         IF (sdbfrz:=0) THEN frzblk(sdbblk, -1);
         IF (destfrz:=0) THEN frzblk(destblk, -1);
         DROP (sigrtn);
      RETURN( TRUE );
      (rtnf3):                                                   10D3Q
      DROP (sigrtn);
      IF (sdbfrz:=0) THEN frzblk(sdbblk, -1);
      IF (destfrz:=0) THEN frzblk(destblk, -1);
      RETURN (FALSE);
      END.

   (delprop) %xxx delete property %
   PROCEDURE ( stid, proptyp );                                   10D4
      % deletes the property block and any associated inferior tree
      structure  for the block proptyp block of the indicated node.
      Returns TRUE if successful, 0 if not. %
      %---------------%
      LOCAL sdb, stdb;
      LOCAL sdbit; % stid of inferior tree of property freed; it
      must be freed as well %
      REF sdb;
      INVOKE (sigrtn, rtnf4);
      IF NOT lodprop( stid, proptyp : &sdb, stdb) THEN
         RETURN(0);
      IF (sdbit _ freprop( stdb)) THEN
         freintree( sdbit );
      DROP (sigrtn);
      RETURN( TRUE );
      (rtnf4):                                                   10D4K
      DROP (sigrtn);
      RETURN (FALSE);
      END.

   (reprop) % xxx  replace property %
   PROCEDURE (stid, proptyp, length,  data);                      10D5
      % replaces the property block indicated by stid and proptyp
      with a block with data as indicated.  If length is the same as
      the length of data in existing property block, a short cut may
      be taken and the data overwrites the old data.  If, however,
      the length is different, a new block is built and linked in.
      The inferior tree is not replaced in any case:  it remains the
      same.  The inferior tree's pointer to the "owning" property
      block is changed.  Uses filesc if this is a text block.  %
      %---------------%
      LOCAL
         sdb,
         sdbblk,
         stdb,
         sdbit; % stid of inferior tree of freed property; will be
         linked into new property %
```

```
REF data, sdb;
INVOKE (sigrtn, rtnf5);
IF proptyp = txttyp THEN
   BEGIN
   *sar* _ *[&data-1]*;
   rplsid _ stid;
   filesc();
   DROP (sigrtn);
   RETURN( TRUE );
   END;
% get the block to be replaced.  If there is none, this is an
error %
   IF NOT (sdbblk _ lodprop( stid, proptyp : &sdb, stdb)) THEN
      BEGIN
      DROP (sigrtn);
      RETURN(FALSE);
      END;
% if the length is different, delete the block and create a
new one.  Otherwise, simply copy the data into the correct
location.  (Since text handled elsewhere this simplification
works! %
   IF (sdb.slength-sdbhdl) = length THEN
      BEGIN
      frzblk( sdbblk, 1);
      INVOKE (frz1, frzrtn);
      sdb.sinit _ cinit;
      sdb.stime _ gtadcall(); % get date and time %
      IF &data THEN mvbfbf( &data, &sdb+sdbhdl, length);
      frzblk( sdbblk, -1);
      (frzrtn):                                          10D5H1H
      DROP (frz1);
      DROP (sigrtn);
      RETURN( TRUE );
      END
   ELSE
      BEGIN
      sdbit _ freprop( stdb );
      IF (stdb _ crepr2( stid, proptyp, length, &data, FALSE,
      0, 0)) THEN
         BEGIN
         % link old inferior tree to new data block %
            insitree( sdbit, stdb );
         END;
      DROP (sigrtn);
      RETURN( stdb % TRUE if new block, FALSE if not % );
      END;
(rtnf5):                                                 10D5I
DROP (sigrtn);
RETURN (FALSE);

(frz1) CATCHPHRASE;                                      10D5M
   BEGIN
   frzblk(sdbblk, -1);
   TERMINATE;
   END;
```

```
        END.

% Inferior tree %
    (creit) %xxx create inferior tree %
    PROCEDURE (stid, proptyp);                                    10E1
        % Creates the origin of an inferior tree and links it to the
        data block property specified by stid and proptyp.  Returns 0
        if error or stid of origin of inferior tree. %
        %---------------%
        LOCAL sdb, stdb, newstid, ring;
        REF sdb, ring;
        IF NOT lodprop (stid, proptyp : &sdb, stdb) THEN
           RETURN(0);
        newstid _ newrng( stid.stfile: &ring);
        % Initialize inferior tree origin fields %
           ring.rsuc _ ring.rsub _ newstid.stpsid;
           ring.rhf _ ring.rtf _ ring.rtorgin _ TRUE;
           % rnameh and rnamef and rsid have been set in newrng.
           Other fields were also zeroed. %
        insitree(newstid, stdb);
        RETURN(newstid);
        END.

    (copit) % xxx copy inferior tree%
    PROCEDURE ( stid, proptyp, destid );                          10E2
        % copies inferior tree of property block at node indicated by
        stid and proptyp to the proptyp block of destid.  Returns TRUE
        if successful, 0 if error %
        %---------------%
        LOCAL sdb, stdb, stidit;
        REF sdb;
        INVOKE (sigrtn, rtnf6);
        IF NOT lodprop(stid, proptyp : &sdb, stdb ) THEN
           BEGIN
           DROP (sigrtn);
           RETURN( FALSE );
           END;
        IF NOT stidit _ sdb.sitpsid THEN
           BEGIN
           DROP (sigrtn);
           RETURN( TRUE );
           END;
        stidit.stfile _ stid.stfile;
        IF NOT lodprop(destid, proptyp : &sdb, stdb ) THEN
           BEGIN
           DROP (sigrtn);
           RETURN( FALSE );
           END;
        IF NOT trecop( stidit, stdb ) THEN
           BEGIN
           DROP (sigrtn);
           RETURN( FALSE );
           END;
        DROP (sigrtn);
        RETURN( TRUE );
        (rtnf6):                                                  10E2M
```

```
      DROP (sigrtn);
      RETURN (FALSE);
      END.

   (movit) % xxx move inferior tree%
   PROCEDURE ( stid, proptyp, destid);                             10E3
      % moves the inferior tree associated with property block
      indicated by stid and  proptyp to  the property block proptyp
      associated with node destid  Returns true if OK, 0 if error %
      %---------------%
      LOCAL sdb, stdb, stidit;
      REF sdb;
      INVOKE (sigrtn, rtnf7);
      IF NOT lodprop( stid, proptyp: &sdb, stdb ) THEN
         BEGIN
         DROP (sigrtn);
         RETURN( FALSE );
         END;
      IF NOT (stidit _ sdb.sitpsid := 0) THEN
         BEGIN
         DROP (sigrtn);
         RETURN( TRUE ); % No inferior tree to move %
         END;
      stidit.stfile _ stid.stfile;
      IF NOT lodprop( destid, proptyp: &sdb, stdb ) THEN
         BEGIN
         % release the inferior tree %
            freintree( stidit );
         DROP (sigrtn);
         RETURN( FALSE );
         END;
      IF stid.stfile = destid.stfile THEN
         insitree( stidit, stdb )
      ELSE
         BEGIN
         % Copy & link in inferior tree to new file %
            IF NOT trecop( stid, stdb ) THEN
               BEGIN
               DROP (sigrtn);
               RETURN( FALSE );
               END;
         % Release the old inferior tree %
            freintree( stidit );
         END;
         DROP (sigrtn);
      RETURN( TRUE );
      (rtnf7):                                                     10E3M
      DROP (sigrtn);
      RETURN (FALSE);
      END.

   (delit) %xxx delete inferior tree%
   PROCEDURE (stid, proptyp );                                     10E4
      % Deletes the inferior tree of the indicated property block.
      Unlinks it and releases space. Returns True if successful, 0
      if not. %
```

```
        %---------------%
        LOCAL sdb, stdb, stidit;
        REF sdb;
        INVOKE (sigrtn, rtnf8);
        IF NOT lodprop( stid, proptyp : &sdb, stdb) THEN
           BEGIN
           DROP (sigrtn);
           RETURN( FALSE );
           END;
        IF NOT (stidit _ sdb.sitpsid) THEN
           BEGIN
           DROP (sigrtn);
           RETURN(TRUE) % none to delete %
           END
        ELSE stidit.stfile _ stid.stfile;
        freintree( stidit );
        DROP (sigrtn);
        RETURN( TRUE );
        (rtnf8):                                                    10E4L
        DROP (sigrtn);
        RETURN (FALSE);
        END.

% basic entity structural edit support %
   % edit support %
      (freprop) % xxx ***% PROCEDURE (stdb);                        11A1
        % Frees property, but NOT associated inferior tree!  The
        calling procedure must explicitly call freintree with the stid
        of the  inferior tree to have that done.  (This permits
        replacement of data without removing the inferior tree.)  To
        facilitate that, this procedure returns: FALSE if there is no
        inferior tree or the stid of the inferior tree if there is
        one. %
        LOCAL
           blknum, % index into CRPGAD fo this block %
           blkad,  %address of block containing the SDB%
           sdbit, % psid of inferior tree; if non-zero, we must
           release the inferior tree including all its structure and
           data elements. %
           dt,     %pointer to DTBST entry for the block%
           blkfre, %address of free space for the block%
           sdbblk, %address of SDB to be freed%
           sdbpt,  %pointer to SDB's in the block%
           savstdb, % stdb of first db in block of stdb (stwc field
           zeroed) %
           fileno, % of file in which stdb lives %
           blkfrz,
           blknxt; %another pointer to SDB's in the block%
        REF dt;
        blkfrz _ 0;
        IF stdb.stpsdb = 0 THEN RETURN( FALSE );
        % get block to be freed into core; sdbblk will be its address
        in core and blkad will be the address of the start of the page
        in which it lives. %
           % Set stwc field of savstdb copy of stdb  to zero so that
           lodsdb  will return as its second parameter the address in
      core of the first word in the page containing the data
      block.  Get the address  of the data block by adding wc
      back in. %
      fileno _ stdb.stfile;
      savstdb _ stdb;
      savstdb.stwc _ 0;
      dblst _ stdb.stblk; % save in global for use by newdb %
      blknum _ lodent(savstdb, sdbtyp :blkad);
      frzblk( blknum, 1);
      blkfrz _ TRUE;
      INVOKE (sigfrz);
      sdbblk _ blkad + stdb.stwc;
   % save away  pointer to inferior tree.  we must delete it as
   well.  We do it after this block is unfrozen below. %
      IF (sdbit _ [sdbblk].sitpsid) THEN
         sdbit.stfile _ fileno;
   % Make PC if necessary; calls err if trouble, resets if
   catastrophe %
      crepc(fileno);
   % Unlink the block  Does not delete inferior tree.%
       IF NOT unlnkprop( sdbblk, stdb) THEN
         BEGIN
         err($"System error while unlinking data block.  Possible
         bad file.");
         END;
   %pointer to the dtbst entry%
      &dt _ filehead[fileno] + $dtbst - $filhed + stdb.stblk;
   %record decrease in used count%
      dt.rfused _ dt.rfused - [sdbblk].slength;
   %pointer to free space%
      % get absolute location of start of free block for this
      page %
      blkfre _ dt.rffree + blkad;
      % get absolute core address of first sdb in page %
      sdbpt _ blkad + fbhdl;
      IF sdbblk > sdbpt THEN
         %the block to be freed is not the first%
         %merge the preceeding SDB if it is garbage%
         BEGIN
         %move sdbpt to the SDB in front of the one to be
         freed%
         LOOP CASE blknxt _ sdbpt + [sdbpt].slength OF
            = sdbblk : EXIT;
            <= sdbpt : badfil(fileno);
            ENDCASE sdbpt _ blknxt;
         %if it is garbage, then merge%
         IF [sdbpt].sgarb THEN
            BEGIN
            [sdbpt].slength _ [sdbpt].slength +
            [sdbblk].slength;
            sdbblk _ sdbpt;
            END;
         END;
      [sdbblk].sgarb _ TRUE;
      IF (blknxt _ [sdbblk].slength + sdbblk) = blkfre THEN
         %can add to free space%
```

```
            dt.rffree _ sdbblk - blkad
         ELSE IF [blknxt].sgarb THEN
            %merge the two%
            [sdbblk].slength _ [sdbblk].slength +
            [blknxt].slength;
      % Unfreeze frozen block %
         IF (blkfrz:=0) THEN frzblk(blknum, -1);
         DROP (sigfrz);
      RETURN( sdbit );  % the inferior tree stid: it has not been
      released! %

      (sigfrz) CATCHPHRASE;                                      11A1P
         BEGIN
         DISABLE (sigfrz);
         IF (blkfrz:=0) THEN frzblk(blknum, -1);
         CONTINUE;
         END;
      END.

   (crepr2) % xxx   Create property block %
   PROCEDURE ( stid, proptyp, length, data, initflg, dlleft, dlright
   );                                                             11A2
      % Builds a data block of property type proptyp (which must be
      a valid type atssigned (and declared ) by ARC and links it
      into the plist associated with the stid in the proper order
      (determined in the procedure locprop).  If such a property
      already exists in the node, we have an error:  it must first
      be deleted.  Returns stdb of new block or 0 if error.  length
      is the length of the data and data is a pointer to an array of
      length words in which the data is stored. If initflg is TRUE,
      use the values in dlleft, dlright for the name delimiters;
      otherwise compute them if this is a text block.  If &data is
      zero, do not copy data; just initialize the block. %
      %---------------%
      LOCAL
         destblk, % page number with destination (ring or sdb) %
         desel, % address of destination ring or sdb %
         dest, %stdb or stid of destination %
         destrng, % TRUE if destination a ring, FALSE if a property
         %
         nwblk, % page with new db %
         sdbnew, % address of new db %
         nwtdb, % stdb of new sdb %
         fhdloc, % address of the file header %
         destfrz, nwfrz,
         rnl, % address of the ring element %
         upstid; % stid of up used in calculating name delimiters %
      LOCAL STRING escname[100];
      REF data, desel, sdbnew, rnl;
      % Get and freeze destination; locprop finds the proper
      location for this new block. %
         IF NOT (destblk _ locprop(stid, proptyp : &desel, dest,
         destrng)) THEN
            BEGIN
            RETURN(FALSE);
            END;
         frzblk( destblk, 1);
         nwfrz _ 0;
         destfrz _ TRUE;
         INVOKE (sigrtn, rtnf9);
      % Get a new block %
         nwtdb _ newdb(length+sdbhdl, proptyp, dest.stfile :
         &sdbnew, nwblk);
         frzblk( nwblk, 1);
         nwfrz _ TRUE;
      % copy data from buffer to new block if requested %
         IF &data THEN mvbfbf( &data, &sdbnew+sdbhdl, length);
      % link in the property. lnkprop assumes appropriate blocks are
      in core and frozen.  locprop has been used to find the correct
      location for the new property block %
         lnkprop( destrng, &desel, dest, &sdbnew, nwtdb);
      % set name and other text property dependent fields %
         IF proptyp = txttyp THEN
            BEGIN
            % Initialize the name delimiters to be assigned to the
            statement. %
               IF NOT initflg THEN
                  BEGIN
                  % Must calculate the delimiters %
                  IF stid.stpsid = origin THEN
                     BEGIN %use standard delimiters for that file%
                     fhdloc _ filehead[stid.stfile] - $filhed;
                     sdbnew.slnmdl _ [fhdloc + $namdl1];
                     sdbnew.srnmdl _ [fhdloc + $namdl2];
                     END
                  ELSE
                     BEGIN %use same delimiters as up unless up is
                     origin of inferior tree%
                     upstid _ getup( stid );
                     IF getorf( upstid ) THEN
                        BEGIN %use standard delimiters for that
                        file%
                        fhdloc _ filehead[stid.stfile] - $filhed;
                        sdbnew.slnmdl _ [fhdloc + $namdl1];
                        sdbnew.srnmdl _ [fhdloc + $namdl2];
                        END
                     ELSE
                        sdbnew.slnmdl _ getnmdl( upstid :
                        sdbnew.srnmdl);
                     END;
                  END
               ELSE
                  BEGIN
                  % Use the old values passed as parameters. %
                  sdbnew.slnmdl _ dlleft;
                  sdbnew.srnmdl _ dlright;
                  END;
            sdbnew.schars _ [&data-1].L; % assume data non zero for
            text! %
            % check for name %
               stcwrk _ stid;
               stcwrl _ 1;
            fechcl( forward, $stcwrk );
            xtrnam( $escname, $stcwrk, sdbnew.slnmdl,
            sdbnew.srnmdl );
         sdbnew.sname _ IF escname.L = empty THEN 1 ELSE stcwr1;
         IF destrng THEN
            BEGIN
            IF escname.L = empty THEN
               BEGIN
               desel.rnamef _ desel.rnameh _ 0;
               END
            ELSE
               BEGIN
               desel.rnameh _ hash($escname);
               desel.rnamef _ TRUE;
               END;
            END
         ELSE
            BEGIN
            lodent(  stid, rngtyp : &rnl );
            IF escname.L = empty THEN
               BEGIN
               rnl.rnamef _ rnl.rnameh _ 0;
               END
            ELSE
               BEGIN
               rnl.rnameh _ hash($escname);
               rnl.rnamef _ TRUE;
               END;
            END;
         END;
   % Unfreeze blocks %
      IF (destfrz:=0) THEN frzblk(destblk, -1);
      IF (nwfrz:=0) THEN frzblk(nwblk, -1);
   DROP (sigrtn);
   RETURN( nwtdb );
   (rtnf9):                                                    11A2N
   DROP (sigrtn);
   IF (destfrz:=0) THEN frzblk(destblk, -1);
   IF (nwfrz:=0) THEN frzblk(nwblk, -1);
   RETURN (FALSE);
   END.

(insitree) %***%                                               11A3
   %Given the stid of the origin of an inferior tree, this
   routine conects it to the data block passed as the second
   argument.  Assumes other fields. have been set up already.
   Cf. TRECOP and CREIT.%
   %-------------%
   PROCEDURE(stid, nwtdb);
   stosdb( stid, nwtdb.stpsdb);
   stoitree( nwtdb, stid.stpsid);
   RETURN;
   END.

(freintree) %***% PROCEDURE (sdbit);                           11A4
   % free inferior tree:  release all structure blocks and their
```

```
      associated data blocks in the inferior tree the origin of
      which  is sdbit %
      % this ought to work; but should the correspondence table have
      endfil?  (remgrp is not necessary.  this has been done already
      in freprop) %
         delgrp( sdbit, sdbit, endfil);
         % delgrp calls dsttx.  This should, in fact, call freplist
         to get rid of them all!! %
      RETURN;
      END.

%structural inserts%

   (newsuc)                                                     11B1
      %This routine gets a new stid and then inserts it as the suc
      of the stid passed it.  It returns the new stid.%
      %-------------%
      PROCEDURE(stid);
      LOCAL newstd; %new stid%
      newstd _ newrng(stid.stfile);
      inss(stid, newstd, newstd);
      RETURN(newstd);
      END.

   (newsub)                                                     11B2
      %This routine gets a new stid and then inserts it as the sub
      of the stid passed it.  It returns the new stid.%
      %-------------%
      PROCEDURE(stid);
      LOCAL newstd; %new stid%
      newstd _ newrng(stid.stfile);
      insd(stid, newstd, newstd);
      RETURN(newstd);
      END.

   (insgrp)                                                     11B3
      %Insert SRC1, SRC2 at TARGET according to DIR.  (SRC1 and SRC2
      are assumed to define a legal, ordered group.)%
      %-------------%
      PROCEDURE(target, dir, src1, src2);
      target _ rlevset(target, dir : dir);
      IF (target.stpsid # origin AND NOT getorf( target )) AND dir =
      levsuc THEN
         inss(target, src1, src2)
      ELSE insd(target, src1, src2);
      RETURN;
      END.

   (inss)                                                       11B4
      %Given an stid, this routine inserts a group, defined by the
      second and third arguments, as the suc of the first argument.
      First it makes the suc of STID the suc of GRP2, and updates
      the tail flag; then it makes GRP1 the suc of STID and updates
      the head and tail flags.%
      %-------------%
      PROCEDURE(stid, grp1, grp2);
      IF stid.stfile # grp1.stfile THEN err($"illegal insert");
      IF grp1.stfile # grp2.stfile THEN err($"illegal group");
      stosuc(grp2, getsuc(stid));
      stoftl(grp2, getftl(stid));
      stosuc(stid, grp1);
      stofhd(grp1, FALSE);
      stoftl(stid, FALSE);
      RETURN;
      END.

   (insd)                                                        11B5
      %Given an stid, this routine inserts the group defined by the
      second and third arguments down from the first argument.%
      %-------------%
      PROCEDURE(stid, grp1, grp2);
      LOCAL substd; %stid of sub of stid passed as argument%
      IF stid.stfile # grp1.stfile THEN err($"illegal insert");
      IF grp1.stfile # grp2.stfile THEN err($"illegal group");
      IF (substd _ getsub(stid)) # stid THEN
         BEGIN
         stofhd(substd, FALSE);
         stoftl(grp2, FALSE);
         END
      ELSE stoftl(grp2, TRUE);
      stosuc(grp2, substd);
      stosub(stid, grp1);
      stofhd(grp1, TRUE);
      RETURN;
      END.

   (rlevset)                                                     11B6
      % Determines target stid and direction for inserting
      statements.  Given an stid and a relative levadj count,
      returns an stid and levdown if to be inserted down, levsuc if
      to be inserted as successor to returned stid. %
      %---------------%
      PROCEDURE(stid, dir);
      LOCAL count, numb;
      CASE dir OF
         =0: dir _ levsuc; %successor%
         <0: dir _ levdown; %down%
         ENDCASE %up number of levels indicated by dir%
            WHILE dir > 0 DO
               BEGIN
               stid _ getup(stid);
               dir _ dir - 1;
               END;
      RETURN(stid, dir);
      END.

   (levset)                                                      11B7
      % Determines target stid and direction for inserting
      statements.  Given an stid and the address of a string
      containing u's and d's (a levadj string), returns an stid and
      -1 if to be inserted down, 0 if to be inserted as successor to
      returned stid. %
      %---------------%
      PROCEDURE(stid, levstg);
      LOCAL dir, count, numb;
      REF levstg;
      dir _ 0;
      IF levstg.L # empty THEN
         BEGIN
         count _ 1;
         DO
            CASE *levstg*[count] OF
               ='U, ='u: BUMP DOWN dir;
               ='D, ='d: BUMP dir;
               ENDCASE NULL
         UNTIL (count _ count + 1) > levstg.L;
         END;
      CASE dir OF
         =0: BUMP dir; %successor%
         >0: dir _ 0 %down%
         ENDCASE %up number of levels indicated by dir%
            WHILE (dir _ dir + 1) <= 0
               DO stid _ getup(stid);

      RETURN(stid, dir - 1);
      END.

% locate and link property %
   (locprop) % xxx  locate destination of new property %
   PROCEDURE (destid, proptyp);                                    11C1
      % locates the place after which a new property is to be
      inserted.  Returns four items: first is FALSE if error, page
      number in core if success; second is address of block in core
      (which must be frozen if you want to do anything with it!),
      third is stid or stdb of ring or property block, fourth is
      flag: TRUE if block is ring, false if DB %
      %---------------%
      LOCAL
         destblk,
         desel,
         dest,
         destrng,
         stdb,
         ptabin,
         blknum,
         db;
      REF db, desel;
      IF NOT (destblk _ lodent(destid, rngtyp : &desel)) THEN
         RETURN( FALSE, 0, 0, 0);
      IF NOT (stdb _ desel.rsdb) THEN
         RETURN( destblk, &desel, destid, TRUE);
      stdb.stfile _ destid.stfile;
      dest _ destid;
      destrng _ TRUE;
      IF NOT (ptabin _ getptab(proptyp)) THEN
         RETURN( FALSE, 0, 0, 0); % illegal property type %
      LOOP
         BEGIN
         IF NOT (blknum _ lodent(stdb, sdbtyp : &db)) THEN
            RETURN( FALSE, 0, 0, 0);
         CASE getptab( db.sptype ) OF
            <= 0: RETURN( FALSE, 0, 0, 0); % illegal property type %
            < ptabin:
               BEGIN
               destblk _ blknum;
               &desel _ &db;
               dest _ stdb;
               destrng _ FALSE;
               IF (stdb.stpsdb _ db.spsdb) = 0 THEN
                  REPEAT CASE(ptabin+1); % Force return %
               % Repeat loop %
               END;
            = ptabin: % Property already exists %
               RETURN( FALSE, 0, 0, 0);
            ENDCASE % > ptabin %
               RETURN( destblk, &desel, dest, destrng );
         END;
      END.

   (getptab) % xxx get property table index %
    PROCEDURE (proptyp);                                            11C2
      % Given a property type, this procedure returns its index in
      the property table.  This is the number beyond which we need
      not search. If illegal proptyp (not in table) return FALSE. %
      LOCAL count;
      FOR count _ 1 UP UNTIL > proptab DO
         IF proptab[count] = proptyp THEN RETURN(count);
      IF proptyp IN [40000B, 77777B] THEN RETURN(proptyp);
         %these types reserved for users%
      RETURN(FALSE);
      END.

   (lnkprop)  % xxx   link a property into list structure %
   PROCEDURE( destrng, % TRUE if dest a ring; FALSE if DB %         11C3
      desel, % address of destination RING or DB; frozen %
      dest, % stid or stdb of destination %
      sdbnew, % address of new DB; frozen %
      nwtdb % stdb of new block% );
      REF desel, sdbnew;
      % Link the new block to the destination block %
         % Fill in the SPSDB or RSDB field of the dest block with
         the STPSDB of the new one %
            IF destrng THEN
               BEGIN
               sdbnew.spsdb _ desel.rsdb := nwtdb.stpsdb;
               sdbnew.spsid _ dest.stpsid;
               END
            ELSE
               BEGIN
               sdbnew.spsdb _ desel.spsdb := nwtdb.stpsdb;
               sdbnew.spsid _ desel.spsid;
               END;
      RETURN;
      END.
```

```
(unlinkprop) % xxx  unlink a property from list structure %
PROCEDURE( sdbblk, % address of DB to be unlinked; assumed frozen
in core %                                                     11C4
   stdb % of DB to be unlinked% );
   LOCAL
      stid,       %of node; one of its properties is being freed.
      %
      sdbprop, % value of next property field. %
      rnl,        %address of ring block coresponding to stid. %
      svstdb, %STDB of a property in the node %
      fileno, % file in which block lives %
      nxtblk; % sdb corresponding to svstdb %
   REF sdbblk, rnl, nxtblk;
   IF stdb.stpsdb = 0 THEN RETURN( FALSE);
   fileno _ stdb.stfile;
   % link up other properties in this node around the deleted
   one. %
      stid _ sdbblk.spsid;
      stid.stfile _ fileno;
      sdbprop _ sdbblk.spsdb;
      lodent(stid, rngtyp : &rnl);
      % Check if text block is being freed; if so, 0 name fields
      in ring %
         IF sdbblk.sptype = txttyp THEN
            BEGIN
            rnl.rnamef _ rnl.rnameh _ 0;
            END;
      IF (svstdb _ rnl.rsdb) # stdb.stpsdb THEN
         BEGIN
         LOOP % over properties in this node %
            BEGIN
            svstdb.stfile _ fileno;
            lodent(svstdb, sdbtyp : &nxtblk);
            IF (svstdb _ nxtblk.spsdb) = stdb.stpsdb THEN
               BEGIN
               nxtblk.spsdb _ sdbprop;
               EXIT LOOP;
               END;
            IF svstdb = 0 THEN EXIT LOOP;
            END;
         END
      ELSE
         BEGIN
         rnl.rsdb _ sdbprop;
         END;
   RETURN( TRUE );
   END.

(copnam) % xxx copy name fields from existing to new property %
PROCEDURE( destrng, % TRUE if dest a ring; FALSE if DB %      11C5
   sdbold, % address of source DB; assumed frozen in core %
   oldstid, % source stid %
   desel, % address of destination RING or DB; frozen %
   dest % stid or stdb of destination % );
   LOCAL
```

```
            newstid, % stid of node to which new block is attached %
            oldnam, % name hash of source stid %
            oldrng,
            newring;
         REF sdbold, desel, oldrng, newring;
         % set name hash if necessary %
            IF sdbold.sptype = txttyp THEN
               BEGIN
               % Must copy name hash to ring %
               lodent( oldstid, rngtyp : &oldrng);
               oldnam _ oldrng.rnameh;
               IF destrng THEN
                  BEGIN
                  desel.rnamef _ oldnam # 0;
                  desel.rnameh _ oldnam;
                  END
               ELSE
                  BEGIN
                  % Must load ring; text not necessarily first %
                  newstid _ desel.spsid;
                  newstid.stfile _ dest.stfile;
                  lodent(newstid, rngtyp : &newring);
                  newring.rnamef _ oldnam # 0;
                  newring.rnameh _ oldnam;
                  END;
               END;
         RETURN;
         END.

   % copy tree %
      (trecop) % xxx ***% %Copies a property list.  Arguments are    11D1
         1) STID (includes STFILE field -- source file) of inferior
         tree to be copied,
         2) STDB (includes STFILE field -- destination file) of new
         block to which it is to be attached.  Returns stid of origin
         of inferior tree if OK; FALSE if not.
         %

         PROCEDURE (itstid, nwtdb);
         LOCAL
            oldsid,
            newsid,
            ring;
         REF ring;
         % what happens if we are doing copy filtered?  if a node
         passes the filter, do we copy everything in the inferior tree
         associated with that node?  what to do? %
         INVOKE (sigrtn, rtnf10);
         IF NOT getorf( itstid ) THEN
            BEGIN
            DROP (sigrtn);
            RETURN( FALSE );
            END;
         oldsid _ itstid;
         newsid _ newrng(nwtdb.stfile : &ring);
         % Initialize inferior tree origin fields %
         ring.rsuc _ ring.rsub _ newsid.stpsid;
         ring.rhf _ ring.rtf _ ring.rtorgin _ TRUE;
         % rnameh and rnamef and rsid have been set in newrng.
         Other fields were also zeroed. %
      LOOP
         BEGIN
         %get stid's, this branch%
         WHILE (oldsid := getsub(oldsid)) # oldsid DO
            newsid _ newsub(newsid);
         %now copy sdb's and point to next branch%
         LOOP
            BEGIN
            copplist(oldsid, newsid);
            upctbl(oldsid, newsid, oldsid);
            IF oldsid = itstid THEN %tree copy complete%
               BEGIN
               % link up the inferior tree to the property data
               block %
               insitree( newsid, nwtdb);
               DROP (sigrtn);
               RETURN( newsid );
               END;
            IF NOT getftl(oldsid := getsuc(oldsid)) THEN EXIT;
               %still more in plex%
            newsid _ getsuc(newsid);
            END;
         newsid _ newsuc(newsid);
         END;
      (rtnf10):                                                   11D1N
      DROP (sigrtn);
      RETURN (FALSE);
      END.

%Ring Utility Routines%

   (newrng) %find room for a new ring element and allocate it.
   Called with file number of file where want the new element.
   Returns                                                        11E1
      1) STID and
      2) the address of the new element.%
      PROCEDURE (fileno);
      LOCAL
         rngblk, %counter for ring blocks%
         rngtry, %possible block for use next%
         pgindx, %page number of where loaded%
         blkad,  %address of block where will allocate%
         fileh,  %location of file header%
         nringl, %number of ring elements in block%
         freep,  %free list pointer in block%
         rngsta, %address of RNGST for the file%
         rngloc, %address of RNGL for the file%
         stid,   %new STID%
         rn;     %pointer into the RNGST for the file%
      REF rn;
      rngsta _ $rngst + fileh _ filehead[fileno] - $filhed;
      rngloc _ fileh + $rngl;
```

```
%first check the ring block from which last allocated
or freed an element (number saved in RNGLST)%
&rn _ rngsta + rnglst;
IF rnglst IN [0,rngm) AND
   rn.rfexis AND rn.rfcore AND rn.rffree THEN
   BEGIN
   stid _
      nwrngb(fileh, rnglst, rn.rfcore, fileno : rngloc);
   RETURN (stid, rngloc);
   END;
%else search for a block%
rngblk _ 0;
rngtry _ -1;
&rn _ rngsta;
DO BEGIN
   IF rn.rfexis AND rn.rffree THEN
      IF rn.rfcore THEN
         BEGIN
         stid _
            nwrngb(fileh, rngblk, rn.rfcore, fileno :
            rngloc);
         RETURN (stid, rngloc);
         END
      ELSE rngtry _ rngblk;
   BUMP &rn;
   END
UNTIL (rngblk _ rngblk+1) > [rngloc];
IF rngtry >= 0 THEN %load one that has room%
   BEGIN
   stid _ 0;
   stid.stfile _ fileno;
   stid.stblk _ rngtry;
   pgindx _ lodent( stid, rngtyp : blkad);
   stid _ nwrngb(fileh, rngtry, pgindx, fileno : rngloc);
   RETURN (stid, rngloc);
   END;
%must allocate a new block%
rngblk _ 0;
&rn _ rngsta;
DO BEGIN
   IF NOT rn.rfexis THEN %will initialize this block%
      BEGIN
      % Make PC if necessary; calls err if error, resets if
      catastrophe-- done here because we will write on the
      file header which does not cause a wrtpsi %
         crepc( fileno );
      pgindx _ lodrfb(rngbas+rngblk, niltyp, fileno);
      blkad _ crpgad[pgindx];
      %finish initialization of the header%
      [blkad].fbtype _ rngtyp;  [blkad].fbind _ rngblk;
      %now set up the status table%
      rn.rfexis _ TRUE;
      rn.rfused _ rn.rffree _ fbhdl;
      rn.rfcore _ pgindx;
      %update RNGL for the file%
      IF rngblk > [rngloc] THEN [rngloc] _ rngblk;
```

```
            %finish the initialization of the block%
            %make a free list%
            %calculate the number of elements that will fit
            in a page%
            nringl _ (blksiz-fbhdl) / ringl;
            freep _ blkad + fbhdl;
            DO BEGIN
               [freep] _ freep - blkad + ringl;
               freep _ freep + ringl
               END
            UNTIL (nringl _ nringl-1) = 1;
            %zero the last one%
            [freep] _ 0;
            stid _ nwrngb(fileh, rngblk, pgindx, fileno :
            rngloc);
            RETURN ( stid, rngloc);
            END;
         BUMP &rn;
         END
      UNTIL (rngblk _ rngblk+1) = rngm;
      %have exhausted the space available for structure%
      err($"structure full") END.

   (nwrngb) %Used by newrng to actually allocate the ring element.
   Generates new SID and stores it in the ring element.  Called with
                                                                  11E2
      1) address of file header - $filhed,
      2) the ring block number,
      3) the page index of the ring block,
      4) the file number.
      It allocates a new ring element from that block and returns
      1) the new statement identifier (STID) and
      2) the address of the new element.%
      PROCEDURE (fileh, rngblk, pgindx, fileno);
      LOCAL
         rn,     %address of ring entry in header%
         freep,  %free list pointer%
         nwrn,   %pointer to new ring block%
         nwrne,  %pointer to end of new ring block%
         stid;   %stid for the new element%
      REF rn;
      %record the block number from which allocating the
      element%
      rnglst _ rngblk;
      &rn _ fileh + $rngst + rngblk;
      %check the free pointer for legality%
      IF (freep _ rn.rffree)
         NOT IN [fbhdl,blksiz) THEN badfil(fileno);
      % Make PC if necessary; calls err if error, resets if
      catastrophe-- done here because we will write on the file
      header which does not cause a wrtpsi %
         crepc( fileno );
      freep _ freep + crpgad[pgindx]; %actual address%
      rn.rffree _ [freep]; %new free pointer%
      rn.rfused _ rn.rfused + ringl; %increase used word count%
      %zero the new ring element%
      nwrne _ (nwrn _ freep) + ringl;
      DO [nwrn] _ 0 UNTIL (nwrn _ nwrn+1) = nwrne;
      % get new SID %
      [freep].rsid _ [fileh + $sidcnt] _ [fileh + $sidcnt]
      + 1;
      %return STID for the new element%
      stid _ 0;
      stid.stfile _ fileno;
      stid.stblk _ rngblk;
      stid.stwc _ freep-crpgad[pgindx];
      [freep].rsub _ stid;
      RETURN (stid, freep);
      END.

   (frerng)  %free the ring element for STID given as argument%
                                                                  11E3
      PROCEDURE (stid);
      LOCAL
         rngloc, %location of the ring element%
         blkad,  %address of file block containing the element%
         pgindx, %index of ring block%
         cnt,     %counter for clearing element%
         pnt,     %pointer for clearing element%
         rn;     %pointer to RNGST entry%
      REF rn;
      %zap swork if necessary%
         IF swork = stid THEN swork _ endfil;
         %this is done for the benefit of fechcl.  ensures that when
         a FIND fails and CCPOS is reset to position before the
         FIND, it will be reset to an existing statement or to the
         NULL string.%
      % Make PC if necessary; calls err if error, resets if
      catastrophe-- done here because we will write on the file
      header which does not cause a wrtpsi %
         crepc( stid.stfile );
      rnglst _ stid.stblk;
      pgindx _ lodent( stid, rngtyp : rngloc);
      blkad _ crpgad[pgindx];
      %clear the element%
      cnt _ ringl;
      pnt _ rngloc;
      DO
         BEGIN
         [pnt] _ 0;
         BUMP pnt;
         END
      UNTIL (cnt _ cnt-1) = 0;
      &rn _ filehead[stid.stfile] + $rngst - $filhed +
      stid.stblk;
      %reduce used word count%
      rn.rfused _ rn.rfused - ringl;
      %add to free list%
      [rngloc] _ rn.rffree;
      rn.rffree _ rngloc - blkad;
      RETURN END.
   (goodrng) PROCEDURE(stid);                                   11E4
      %Returns TRUE iff stid points to a ring element which is in
      use%
      %assumes that the file number in the stid is ok%
      LOCAL rngblk, wc, rn;
      REF rn;
      %split the psid into block number and word count%
      wc _ stid.stwc;
      rngblk _ stid.stblk;
      &rn _ filehead[stid.stfile] + $rngst - $filhed + rngblk;
      %check that block number is legal%
      RETURN(rngblk IN [0,rngm) AND rn.rfexis AND getsid(stid)#0);
      END.

%SDB Utility Routines%

   (newdb) %***% %get a new data block of type BLKTYP.  Arguments
   are                                                          11F1
      1) size of new DB,
      2) type of the data block
      3) file number from which to allocate.
      Loads the new data block and performs general initialization.
      Leaves the block loaded, though unfrozen.
      Returns the STDB of the SDB, the SDB and the index of the core
      page to which it has been loaded, %

      PROCEDURE (room, blktyp, fileno);
      LOCAL
         choi2,  %second choice block%
         choi3,  %third choice block%
         least,  %number of used words in choi3%
         diff,   %excess space in garbage SDB%
         free,   %free space start%
         sdbblk, %index in DTBST%
         dt,     %pointer into DTBST%
         pgindx, %page index for block in core%
         dtbsta, %address of DTBST for the file%
         dtbloc, %address of DTBL for the file%
         blkad,  %address of the block%
         blknum,
         sdb,
         sdbpt,  %pointer to SDB's in the block%
         stdb;   %STDB for the new SDB%
      REF dt;
      % general initialization %
         dtbsta _ $dtbst + dtbloc _ filehead[fileno] - $filhed;
         dtbloc _ dtbloc + $dtbl;
         stdb _ 0;
         stdb.stfile _ fileno;
      % Make PC if necessary; calls err if error, resets if
      catastrophe-- done here because we will write on the file
      header which does not cause a wrtpsi %
         crepc( fileno );
      % Check if room in last used block if it's in core %
         &dt _ dtbsta + dblst;
         IF dblst IN [0,dtbm) AND dt.rfexis AND
      dt.rfcore AND dt.rffree + room <= blksiz THEN
         BEGIN %we won this time%
         stdb.stblk _ dblst;
         stdb.stwc  _ dt.rffree;
         dt.rffree _ dt.rffree + room;
         dt.rfused _ dt.rfused + room;
         % initialize data block fields %
            blknum _ initdb( stdb, room, blktyp : sdb);
         RETURN (stdb, sdb, blknum);
         END;
% We have to work harder.  Look at the blocks which are in
core first (in checking tables, if an out of core block has
space without garbage collection but is not in core,  note
that fact in choi2; if it has space, but must be garbage
collected, note that fact in choi3.) %
   choi2 _ choi3 _ -1;
   least _ blksiz;
   sdbblk _ 0;
   &dt _ dtbsta;
   DO BEGIN
      IF dt.rfexis THEN %block already allocated%
         IF dt.rffree + room <= blksiz THEN
            %room in free space%
            IF dt.rfcore THEN %loaded already%
               BEGIN
               stdb.stblk _ dblst _ sdbblk;
               stdb.stwc _ dt.rffree;
               dt.rffree _ dt.rffree + room;
               dt.rfused _ dt.rfused + room;
               % initialize data block fields %
                  blknum _ initdb( stdb, room, blktyp : sdb);
               RETURN (stdb, sdb, blknum);
               END
            ELSE choi2 _ sdbblk
         ELSE %check if there is room if garbage collect%
            %and page not frozen%
            IF dt.rfused + room <= blksiz AND
               dt.rfused < least AND
               (dt.rfcore = 0 OR corpst[dt.rfcore].ctfroz = 0)
               THEN %new best choice%
               BEGIN
               choi3 _ sdbblk;
               least _ dt.rfused;
               END;
      BUMP &dt;
      END
   UNTIL (sdbblk _ sdbblk+1) > [dtbloc];
%enough free space but not in core%
   IF choi2 >= 0 THEN
      BEGIN
      stdb.stblk _ dblst _ choi2;
      &dt _ dtbsta + choi2;
      stdb.stwc _ dt.rffree;
      dt.rffree _ dt.rffree + room;
      dt.rfused _ dt.rfused + room;
      % initialize data block fields %
```

```
               blknum _ initdb( stdb, room, blktyp : sdb);
            RETURN (stdb, sdb, blknum);
            END;
      %if choi3 then block has enough room but may have to garbage
      collect%
         IF choi3 >= 0 THEN
            BEGIN
            stdb.stblk _ dblst _ choi3;
            lodent (stdb, sdbtyp : blkad);
            &dt _ dtbsta + choi3;
            free _ dt.rffree + blkad;
            sdbpt _ blkad + fbhdl;
            UNTIL sdbpt >= free DO
               %try to find a garbage sdb that is big enough%
               BEGIN
               IF [sdbpt].sgarb AND [sdbpt].slength >= room THEN
                  %will put the new SDB here%
                  BEGIN
                  stdb.stwc _ sdbpt - blkad;
                  IF (diff _ [sdbpt].slength - room) > 0 THEN
                     BEGIN %make excess look like garbage sdb%
                     [sdbpt+room].sgarb _ TRUE;
                     [sdbpt+room].slength _ diff;
                        %depends on sgarb, slength being in first
                        word of header since diff may = 1%
                     END;
                  dt.rfused _ dt.rfused + room;
                  % initialize data block fields %
                     blknum _ initdb( stdb, room, blktyp : sdb);
                  RETURN (stdb, sdb, blknum);
                  END;
               %go to the next SDB%
               IF (sdbpt := sdbpt + [sdbpt].slength) >= sdbpt THEN
                  badfil(fileno);
               END;
            %must garbage collect the block%
            gcol(choi3, fileno);
            stdb.stwc _ dt.rffree;
            dt.rffree _ dt.rffree + room;
            dt.rfused _ dt.rfused + room;
            % initialize data block fields %
               blknum _ initdb( stdb, room, blktyp : sdb);
            RETURN (stdb, sdb, blknum);
            END;
      %have to allocate a block%
         sdbblk _ 0;
         &dt _ dtbsta;
         DO BEGIN
            IF NOT dt.rfexis THEN %will initialize this block%
               BEGIN
               pgindx _ lodrfb(sdbblk+dtbbas, niltyp, fileno);
               blkad _ crpgad[pgindx];
               %finish initialization of block header%
               [blkad].fbind  _ sdbblk;
               [blkad].fbtype _ sdbtyp;
               %now set up the status table entry%
            dt.rfcore _ pgindx;
            dt.rfexis _ TRUE;
            dt.rfused _ dt.rffree _ fbhdl+room;
            %update DTBL for the file%
            IF sdbblk > [dtbloc] THEN [dtbloc] _ sdbblk;
            dblst _ sdbblk;
            stdb.stblk _ dblst;
            stdb.stwc _ fbhdl;
            % initialize data block fields %
               blknum _ initdb( stdb, room, blktyp : sdb);
            RETURN (stdb, sdb, blknum);
            END;
         BUMP &dt;
         END
      UNTIL (sdbblk _ sdbblk+1) = dtbm;
   %have exhausted data blocks%
   err($"data storage full") END.

(initdb) %***% PROCEDURE ( stdb, sdbsiz, blktyp);                    11F2
   LOCAL blknum, sdb, end, sdbpt;
   REF sdb;
   % load the block into core %
      blknum _ lodent( stdb, sdbtyp : &sdb);
   % Initialization of general fields in dbheader by newdb %
      end _ (sdbpt _ &sdb) + sdbhdl;
      DO [sdbpt] _ 0 UNTIL (sdbpt _ sdbpt+1) = end;
      sdb.sgarb _ FALSE;
      sdb.slength _ sdbsiz;
      sdb.sinit _ cinit;
      sdb.stime _ gtadcall(); %get time and date%
      sdb.sptype _ blktyp;
   RETURN( blknum, &sdb);
   END.

(gcol) %***% %called by newsdb to garbage collect the sdb block
whose block number and file number are passed as arguments.%
                                                                     11F3
   PROCEDURE (sdbblk, fileno);
   LOCAL
      pgindx,  %page index for core page holding the block%
      pgfrz,
      blkad,   %address of the block%
      blkfre,  %address of free space in block%
      freewd,  %pointer to first garbage block%
      sdbsiz,  %size of SDB%
      pt,      %pointer to SDB%
      stdb,    %stdb used in fixing up linkages%
      tstsdb,
      savsdb,
      elem,
      stid,    %stid for fix up's%
      dt;     %pointer to DTBST entry for the block%
   REF dt, elem;
   pgfrz _ FALSE;
   % Initialize work stids and stdbs with file number %
      stdb _ tstsdb _ savsdb _ stid _ 0;
```

```
         stdb.stfile _ tstsdb.stfile _  savsdb.stfile _ stid.stfile
         _ fileno;
         stdb.stblk _ tstsdb.stblk _ savsdb.stblk _ sdbblk;
      % get the data block status table entry location. %
         &dt _ filehead[fileno] + $dtbst + sdbblk - $filhed;
      % load the block into core and freeze it %
         pgindx _ lodent( stdb, sdbtyp : blkad);
         frzblk(pgindx, 1);
         pgfrz _ TRUE;
      INVOKE (fblk);
      blkfre _ blkad + dt.rffree; %start of free space%
      freewd _ blkad + fbhdl; %first SDB%
      %move to first garbage sdb%
      UNTIL [freewd].sgarb OR freewd >= blkfre DO
         freewd _ freewd + [freewd].slength;
      % In the following loop, pt will point to the first word of
      the non-garbage block to be moved and freewd will point to the
      first free space location which will be filled up. %
      pt _ freewd;
      WHILE pt < blkfre % the first word in free space for block %
      DO
         BEGIN
         IF (sdbsiz _ [pt].slength) = 0 THEN badfil(fileno);
         IF NOT [pt].sgarb THEN
            BEGIN
            mvbfbf(pt, freewd, sdbsiz); %move up the next good
            sdb%
            % ******* %
            % This code saves the old internal pointer-name (stpsdb)
            of the property block which has just been moved up in
            tstsdb.stpsdb.  It then looks at the places which may
            have pointed to it and replaces the pointer with its new
            pointer name (stdb.stpsdb).  If the block moved was the
            first in a property list, the ring's rsdb field will be
            its old name and will be changed;  if not, we must
            thread through the property list, loading the property
            blocks until we come to the one which points to the one
            just moved.  We change the pointer and exit the loop.
            (If we reach the end of the list we have a bad file:
            nobody pointed to the block!)  One other place could
            point to a property block:  that is the block's inferior
            tree.  If the block has one we change its pointer as
            well. %
            %store new stdb -- correct property chain%
               stid.stpsid _ [freewd].spsid;
               stdb.stwc _ freewd-blkad; % the new stdb %
               tstsdb.stwc _ pt - blkad; % the old stdb %
               lodent( stid, rngtyp : &elem);
               IF elem.rsid = 0 THEN err($"Bad statement
               identifier");
               IF tstsdb.stpsdb = (savsdb.stpsdb _ elem.rsdb) THEN
                  elem.rsdb _ stdb.stpsdb
               ELSE
                  LOOP % this is in the middle of a property list %
                     BEGIN
                     lodent( savsdb, sdbtyp : &elem);
```

```
                    IF (tstsdb.stpsdb = (savsdb.stpsdb_elem.spsdb))
                    THEN
                       BEGIN
                       elem.spsdb _ stdb.stpsdb;
                       EXIT LOOP;
                       END
                    ELSE
                       IF savsdb.stpsdb = 0 THEN badfil(fileno);
                    END;
              % Check for inferior tree -- relink if present %
                 IF (stid.stpsid _ [freewd].sitpsid) THEN
                    BEGIN
                    lodent(stid, rngtyp : &elem);
                    IF elem.rsid=0 THEN
                       err($"Bad statement identifier");
                    IF elem.rtorgin THEN
                       elem.rsdb _ stdb.stpsdb;
                    END;
              % ******* %
              freewd _ freewd + sdbsiz;
              END;
           pt _ pt + sdbsiz;
           END;
        %finally update the status table%
        dt.rffree _ freewd - blkad;
        IF dt.rffree # dt.rfused THEN badfil(fileno);
        IF (pgfrz:=0) THEN frzblk(pgindx, -1);
        DROP (fblk);
        RETURN;

        (fblk) CATCHPHRASE;                                          11F3W
           BEGIN
           DISABLE (fblk);
           IF (pgfrz:=0) THEN frzblk(pgindx, -1);
           CONTINUE;
           END;

        RETURN END.

% Miscellaneous support procedures %
  % Freeze blocks in core %
     (frzrfb) %This routine is called for all freezing and thawing of
     random file blocks.  Arguments are                               12A1
        1) file number,
        2) block number in the file,
        3) a 1 to freeze the block, and a -1 to thaw it.
        RERROR is called if that block is not in core.  Anyone
        who freezes a block must be sure it is thawed -- and
        only once.%

        PROCEDURE (fileno, blkn, fr);
        LOCAL
           pgindex, %file block page counter%
           ct;  %pointer to CORPST entry%
        REF ct;
        pgindex _ 1;
```

```
      &ct _ $corpst + 1;
      DO
         BEGIN
         IF ct.ctfull AND
            ct.ctfile = fileno AND
            ct.ctpnum = blkn THEN
               BEGIN
               IF ct.ctfroz + fr NOT IN [0,7] THEN
                  err($"Block frozen too many times in FRZRFB");
               ct.ctfroz _ ct.ctfroz + fr;
               RETURN;
               END;
         BUMP &ct;
         END
      UNTIL (pgindex _ pgindex+1) > rfpmax;
      err($"Block not found in FRZRFB");
      END.

   (frzblk) %Freeze block given as arguments                        12A2
      1) the index in CRPGAD of the block, and
      2) the 1 or -1 for freeze or thaw.%

      PROCEDURE (pgindx, fr);
      LOCAL ct; REF ct;
      &ct _ $corpst + pgindx;
      IF ct.ctfroz + fr NOT IN [0,7] THEN
         err($"Block frozen too many times in frzblk");
      ct.ctfroz _ ct.ctfroz + fr;
      RETURN END.

* Bad file %
   (badfil) %called when find something screwed up in the file%  12B1
      PROCEDURE (fileno);
      bfilno _ fileno; %save it for SIGNAL%
      ABORT(-5, $"Bad File");
      END.
%correspondence table manipulation%

   (upctbl)                                                          12C1
      %Given three stid's, this routine will update occurrences of
      the first stid, using rplstid as the stid that contains the
      text corresponding to oldsid, and newsid as the stid that is
      the replacement for oldsid.  (rplstid should be endfil if the
      original text has been eliminated.)%
      %-------------%
      PROCEDURE(oldsid, rplstid, newsid);
      LOCAL list, listnd;
      REF list;
      IF clstad = 0 OR [clstad].clbuff = 0 THEN RETURN;
      &list _ [clstad].clbuff;
      listnd _ &list + [clstad].clcnt * cll;
      FOR &list UP cll UNTIL >= listnd
      DO
         IF list.clst1 = oldsid THEN
            BEGIN
            list.clst2 _ newsid;
            list.clst1 _ rplstid;
            END
         ELSE
            IF list.clst2 = oldsid THEN
               list.clst2 _ newsid;
      RETURN;
      END.

%File header location%

   (filhdr) PROCEDURE (fileno);                                      12D1
      %returns the address of the file header for the file whose
      number is passed%
      %--------------%
      LOCAL fl;
      REF fl;
      &fl _ (fileno-1)*filstl + $filst;
      RETURN (crpgad[fl.flhead] + fbhdl) END.

%test and set bounds of structure%

   (grptst)                                                          12E1
      %Given two stid's, this routine checks that they specify a
      legal group; it also returns them ordered (GRP1,GRP2). If the
      stid's do not form a legal group, an err($"illegal group") is
      issued.%
      %-------------%
      PROCEDURE(stid1, stid2);
      LOCAL t1, %working stid1%
         t2; %working stid2%
      IF (stid1.stpsid # stid2.stpsid AND (stid1.stpsid = orgstid OR
      stid2.stpsid = orgstid))
      OR stid1.stfile # stid2.stfile THEN err($"illegal group");
      t1 _ stid1; t2 _ stid2;
      LOOP
         BEGIN %is stid2 on same level, and after, stid1%
         IF t1 = stid2 THEN RETURN(stid1, stid2);
         IF getftl(t1) THEN LOOP
            BEGIN %is stid1 on same level after stid2%
            IF t2 = stid1 THEN RETURN(stid2, stid1);
            IF getftl(t2) THEN err($"invalid group selection");
            t2 _ getsuc(t2);
            END;
         t1 _ getsuc(t1);
         END;
      END.

% write Pseudo interrupt for file pages%

   % write psi %
   (wrpsproc) PROCEDURE;                                             13A1
      (wrtpsi):                                                      13A1A
      % save the accumulators %
      svac1 _ R1; R1 _ $svacs; !BLT R1, svacse;
      S _ S + 400000040B;
      wrpi();
      !HRLZI R1, svacs;
      !BLT R1, 17B;
      R1 _ svacl;
      !JSYS debrk;
      END.

   (wrpi) % write pseudo interrupt routine %                          13A2
      PROCEDURE;
      LOCAL trpw, trpd, tpga, tpgx, fl, ct, rf, topc, flg;
      REF fl, ct, rf;
      drastic _ FALSE;
      % read the trap words %
      R1 _ 4B5; !JSYS gtrpw;
      trpw _ R1; trpd _ R2;
      *lit* _ NULL;
      % make sure that really is a write trap %
      IF NOT trpw .A 4B6 THEN
         BEGIN
         *lit* _ "Bad interrupt";
         [levtab] _ $wpifat;
         END;
      tpga _ trpw .A 777B3;
      FOR tpgx _ 1 UP UNTIL > rfpmax DO
         IF crpgad[tpgx] = tpga THEN
            BEGIN
            % set up pointers %
            &ct _ $corpst + tpgx;
            &fl _ (ct.ctfile-1)*filstl + $filst;
            &rf _ crpgad[fl.flhead] + fbhdl - $filhed;
            &rf _ &rf + $rfbs + ct.ctpnum;
            IF rf.rfpart THEN % page from partial copy %
               BEGIN
               IF fl.flpcread THEN
                  BEGIN
                  *lit* _ "Cannot write on this file";
                  % Message will be put out by abort %
                  [levtab] _ $wpiabt;
                  RETURN;
                  END;
               topc _ FALSE;
               R2 _ 14B10;
               END
            ELSE
               BEGIN % page from original file %
               % check if have a partial copy %
               IF NOT fl.flpart THEN % dont have one %
                  IF (NOT fl.flbrws AND NOT lkfile(&fl)) OR NOT
                  makepc(&fl, ct.ctfile, FALSE, $lit : drastic) THEN
                     BEGIN
                     IF NOT fl.flbrws THEN lkun(&fl, $lit); %change
                     user setable word to unlock%
                     % Message will be put out by abort %
                     [levtab] _ $wpiabt;
                     RETURN END;
               IF fl.flpcread THEN
                  BEGIN
```

```
                    *lit* _ "Cannot write on this file";
                    % Message will be put out by ABORT %
                    [levtab] _ $wpiabt;
                    RETURN;
                    END;
                 topc _ TRUE;
                 rf.rfpart _ TRUE;
                 R2 _ 1004B8;
                 END;
              % change access %
              R1.LH _ 4B5;
              R1.RH _ tpga / 512;
              !spacs(R1);  % 60B %
              %must check if rm,w  or write%
              IF trpw .A 10B6 THEN  %read modify write%
                 BEGIN
                 [trpw.RH] _ [trpw.RH];   %must touch page to make
                 private%
                 END
              ELSE [trpw.RH] _ trpd;  %do write for write only case!%
              % map to pc if needed %
              IF topc THEN
                 BEGIN
                 R1.RH _ tpga/512;
                 R1.LH _ 4B5;
                 R2.RH _ ct.ctpnum;
                 R2.LH _ fl.flpart;
                 !pmap(R1, R2, 14B10);
                 IF tops20flag THEN
                    %tops20 releases the page from the map, get it
                    back%
                    BEGIN
                    !EXCH R1,R2;
                    !pmap();
                    END;
                 END;
              RETURN;
              END;
      % the write is not into a file page %
      *lit* _ "Illegal write at location ", STRING( (levllc - 1) .A
      18M, 8);
      [levtab] _ $wpifat;
      END.

    (wpiabt)PROC;                                                13A3
      % non-fatal error: no write access %
      ABORT( nowrtacc, $lit);
      END.
    (wpifat)PROC;                                                13A4
      % non-fatal error: no write access %
      ABORT( wrtfat, $lit);
      END.
  % lock procedures %
    %moved to IOEXEC%
  % make PC %
    (crepc) % create a PC if necessary%
```

```
   PROCEDURE( fileno );                                              13C1
      % Used by procedures which write on the file header page to
      create PC because they do not go through the wrtpsi mechanism.
       Calls err if trouble.  If catastrophe, does a dismes and
      resets NLS. %
      LOCAL fl;
      REF fl;
      IF NOT filepart[fileno] THEN %Make pc if neccessary%
         BEGIN
         &fl _ (fileno-1)*filstl + $filst;
         IF (NOT fl.flbrws AND NOT lkfile(&fl)) OR
            NOT makepc(&fl, fileno, FALSE, $lit : drastic) THEN
               BEGIN
               IF drastic THEN
                  BEGIN
                  % catastrophic error; reset system %
                  *lit* _ "Fatal error in PC creation:  ", *lit*;
                  ABORT( wrtfat, $lit);
                  END;
               IF NOT fl.flbrws THEN
                  lkun(&fl, $lit); %change user setable word to
                  unlock%
               err($lit);
               END;
         END;
      RETURN;
      END.

   %  makepc moved to IOEXEC  %

FINISH of filmnp
```

```
(alldsp)        <nine, frontend, 01970>        PROCEDURE              8B
(askuser)       <nine, frontend, 04952>        PROCEDURE              13A
(auxchr)        <nine, frontend, 02316>        PROCEDURE              15B
(auxinterminate) <nine, frontend, 02371>       PROCEDURE              15C
(auxstartup)    <nine, frontend, 02285>        PROCEDURE              15A
(bbound)        <nine, frontend, 04577>        EXT CONSTANT =1        4B
(ckrrings)      <nine, frontend, 03218>        PROC                   18A
(cmdfinish)     <nine, frontend, 01800>        PROCEDURE              5A
(combnd)        <nine, frontend, 04059>        PROCEDURE              11F
(copyda)        <nine, frontend, 03635>        PROCEDURE              16L
(copysrring)    <nine, frontend, 03436>        PROCEDURE              18K
(copywa)        <nine, frontend, 04436>        PROCEDURE              16M
(curvsp)        <nine, frontend, 02072>        PROCEDURE              9A
(delda)         <nine, frontend, 04233>        PROCEDURE              11G
(delwa)         <nine, frontend, 04388>        PROCEDURE              11H
(dsparea)       <nine, frontend, 02930>        PROCEDURE              16H
(filusd)        <nine, frontend, 01936>        PROCEDURE              6B
(findwa)        <nine, frontend, 02936>        PROCEDURE              16I
(freetrring)    <nine, frontend, 03254>        PROCEDURE              18C
(freesrring)    <nine, frontend, 03422>        PROCEDURE              18J
(fretint)       <nine, frontend, 01886>        PROCEDURE              6A
(frrlength)     <nine, frontend, 03336>        PROCEDURE              18F
(frzchk)        <nine, frontend, 02054>        PROCEDURE              8E
(hinc0)         <nine, frontend, 01797>        EXT CONSTANT =10       4F
(hinc1)         <nine, frontend, 04581>        EXT CONSTANT =14       4G
(hinc2)         <nine, frontend, 04582>        EXT CONSTANT =18       4H
(hinc3)         <nine, frontend, 04583>        EXT CONSTANT =22       4I
(howformat)     <nine, frontend, 02041>        PROCEDURE              8D
(lbound)        <nine, frontend, 04578>        EXT CONSTANT =2        4D
(iccsp)         <nine, frontend, 02824>        PROCEDURE              16B
(icda)          <nine, frontend, 02847>        LOCAL                  16D
(icfile)        <nine, frontend, 02803>        PROCEDURE              16A
(ida)           <nine, frontend, 02855>        PROCEDURE              16F
(mesfre)        <nine, frontend, 01954>        PROCEDURE              7A
(mkdelfrr)      <nine, frontend, 04619>        PROCEDURE              18H
(movbndry)      <nine, frontend, 03914>        PROCEDURE              11E
(newda)         <nine, frontend, 03106>        PROCEDURE              16J
(newtrring)     <nine, frontend, 03242>        PROCEDURE              18B
(newsrring)     <nine, frontend, 03409>        PROCEDURE              18I
(newwa)         <nine, frontend, 03137>        PROCEDURE              16K
(popda)         <nine, frontend, 04850>        PROCEDURE              12C
(pushda)        <nine, frontend, 04819>        PROCEDURE              12B
(pushtrring)    <nine, frontend, 03269>        PROCEDURE              18D
(pushsrring)    <nine, frontend, 03448>        PROCEDURE              18L
(rbound)        <nine, frontend, 04579>        EXT CONSTANT =3        4C
(readfrring)    <nine, frontend, 03296>        PROCEDURE              18E
(readsrring)    <nine, frontend, 03483>        LOCAL                  18N
(recred)        <nine, frontend, 01961>        LOCAL                  8A
(resown)        <nine, frontend, 04411>        PROCEDURE              11I
(rtlast)        <nine, frontend, 04580>        EXT CONSTANT =180      4E
(seldsp)        <nine, frontend, 01998>        PROCEDURE              8C
(setdabnd)      <nine, frontend, 03824>        PROCEDURE              11C
(setwa)         <nine, frontend, 03181>        PROCEDURE              17A
(srrlength)     <nine, frontend, 03525>        PROCEDURE              18O
(storesrring)   <nine, frontend, 04723>        PROCEDURE              18M
(tbound)        <nine, frontend, 01795>        EXT CONSTANT =0        4A
```

```
(uplstnm)      <nine, frontend, 03375>         PROCEDURE              18G
(vinc0)        <nine, frontend, 04584>         EXT CONSTANT =25       4J
(vinc1)        <nine, frontend, 04585>         EXT CONSTANT =30       4K
(vinc2)        <nine, frontend, 04586>         EXT CONSTANT =35       4L
(vinc3)        <nine, frontend, 04587>         EXT CONSTANT =45       4M
(wappend)      <nine, frontend, 03857>         PROCEDURE              11D
(wbpnd)        <nine, frontend, 03739>         PROCEDURE              11B
(wbreak)       <nine, frontend, 03655>         PROCEDURE              11A
(wdelete)      <nine, frontend, 04512>         PROCEDURE              12A
(xfresta)      <nine, frontend, 02179>         LOCAL                  10C
(xrecplasup)   <nine, frontend, 02238>         LOCAL                  14A
(xrelallsta)   <nine, frontend, 02159>         LOCAL                  10B
```

```
< NINE, FRONTEND.NLS;26, >, 17-Mar-78 17:31 KIRK ;;;;  % FRONT END
SUPPORT CODE %
FILE frontend % <ARCSUBSYS>XL10 to <RELNINE>FRONTEND %%
(arcsubsys,xl10,) (RELNINE,frontend.rel,) %
ALLOW!
%compile-time switches%
   SET NSW = FALSE;
%declarations%
   (tbound) EXTERNAL CONSTANT = 0;  %top boundary%                    4A
   (bbound) EXTERNAL CONSTANT = 1;  %bottom boundary%                 4B
   (rbound) EXTERNAL CONSTANT = 3;  %right boundary%                  4C
   (lbound) EXTERNAL CONSTANT = 2;  %left boundary%                   4D
   (rtlast) EXTERNAL CONSTANT = 180;                                  4E
   (hinc0) EXTERNAL CONSTANT = 10;                                    4F
   (hinc1) EXTERNAL CONSTANT = 14;                                    4G
   (hinc2) EXTERNAL CONSTANT = 18;                                    4H
   (hinc3) EXTERNAL CONSTANT = 22;                                    4I
   (vinc0) EXTERNAL CONSTANT = 25;                                    4J
   (vinc1) EXTERNAL CONSTANT = 30;                                    4K
   (vinc2) EXTERNAL CONSTANT = 35;                                    4L
   (vinc3) EXTERNAL CONSTANT = 45;                                    4M
   REF rawchr, msgda, inpt, tda;
%command finish%
   (cmdfinish) PROCEDURE;                                             5A
      REF tda;
      LOCAL srr, frr, stid, cc;  REF frr, srr;
      LOCAL STRING locstr[200];
      IF holdvs = 1 THEN %immed. following mousespecs without refresh%
         BEGIN  %preserve viewspec state until ready to be used%
         holdvs _ 2;
         RETURN;
         END;
      IF cspupdate THEN
         BEGIN
         &tda _ cspupdate;
         stid _ tda.dacsp;
         cc _ tda.daccnt;
         %update statement return ring%
            &frr _ tda.dalink;  %get address of file return ring%
            IF NOT frr.frhexis THEN
               err($"Illegal file return ring detected in cmdfinish");
            %get frr  entry address%
               &frr _ &frr + frrhlen + (frrelen*frr.frhtop);
            IF frr.frexis AND NOT tda.daempty AND tda.dacsp NOT= endfil
            THEN %update srr%
               BEGIN
               %get address of statement return ring%
                  &srr _ frr.frsrring;
               %update old position and viewspecs on ring%
                  storesrring(&srr, 0, tda.dacsp, tda.daccnt,
                  tda.davspec, tda.davspc2);
                     %user may have changed viewspecs%
               END;
         IF curmkr.stfile NOT= tda.dacsp.stfile THEN  %changing files,
         push file return ring%
            BEGIN
         %get name of new file%
            *locstr* _ NULL;
            %-NSW%                                                5A5F2B
               filnam(curmkr.stfile, $locstr);
            %-NSW%                                                5A5F2C
            %+NSW%                                                5A5F2D
               rfilnam(curmkr.stfile, $locstr);
            %+NSW%                                                5A5F2E
         %push new file name on ring%
            pushfrring(tda.dalink, $locstr, curmkr.stfile);
            readfrring(tda.dalink, 0 : &srr);
            IF usesrr THEN  %jump file return -- copy usesrr to new
            srr%
               BEGIN
               copysrring(usesrr, &srr);
               END;
         %put out "modified" message if necessary%
            IF [flntadr(curmkr.stfile)].fllock THEN
               lockmes(curmkr.stfile)
            ELSE dismes(2, $locstr);  %show user new file name%
         %close files no longer used in display areas%
            tda.dacsp _ curmkr;
            tda.daccnt _
               IF nlmode = fulldisplay THEN 1 ELSE curmkr[1];
            tda.daempty _ FALSE;
            freflnt(); %close files no longer needed%
         END;
      tda.dacsp _ curmkr;
      tda.daccnt _
         IF nlmode = fulldisplay THEN 1 ELSE curmkr[1];
      tda.daempty _ FALSE;
      %update viewspecs (most of the time)%
         IF holdvs # 2 THEN
            BEGIN %not next command after mousespec without refresh%
            tda.dapvs _ tda.davspec := cspvs;
            tda.dapvs2 _ tda.davspc2 := cspvs[1];
            END;
      %push new position and viewspecs onto statement return ring%
         IF (NOT (usesrr := 0)) AND (stid NOT= curmkr OR cc NOT=
         curmkr[1]) THEN
            pushsrring(&srr, tda.dacsp, tda.daccnt, tda.davspec,
            tda.davspc2);

      END
   ELSE &tda _ lda();  %set tda to current display area%
   % Set content analyzer and sequence generator in display area %
      IF cspusqcod THEN tda.dausqcod _ cspusqcod := 0;
      IF cspcacode THEN tda.dacacode _ cspcacode := 0;
   IF nlmode = fulldisplay THEN
      BEGIN
      % recreate the display %
         recred();
         cdtype _ dspno; % shut off recred until dpset is called
         again %
      IF lplitreset THEN
         BEGIN
```

```
            litline _ lplitline;
            litreset _ FALSE;
            litapflag _ TRUE;
            rstlit();
            lplitreset _ FALSE;
            END;
         END;
      %update viewspecs (the rest of the time)%
         IF holdvs = 2 THEN
            BEGIN %next command after mousespec without refresh%
            holdvs _ 0;
            tda.dapvs _ tda.davspec := cspvs;
            tda.dapvs2 _ tda.davspc2 := cspvs[1];
            END;
      ckrrings(); %to help find the bug that is smashing the file and
      statement return rings%
      % set up for new command %
         cspupdate _
            IF nlmode = typewriter THEN &tda ELSE 0;
         ecurmkr _ curmkr;  %save for insert statement mode%
         curmkr _ tda.dacsp;
         curmkr[1] _ tda.daccnt;
         cspvs _ tda.davspec;  cspvs[1] _ tda.davspc2;
      % turn off command interrupt (^O) switch %
         [rubmrk] _ FALSE;
      % turn on clear command feed back window switch %
         dspccf _ TRUE ;
      RETURN;
      END.

%.....files in display areas.....%
   (frefint) PROCEDURE;                                                 6A
      %free and close files for files in file status table
      that are't referenced in display table dacsp or frozen
      list%
      %----------------%
      LOCAL fileno, used, fl, da, endfl, endda;
      REF fl, da;
      IF filcnt NOT IN [0, filmax] OR dacnt NOT IN [1, damax]
      THEN
         err($"NLS system error");
      endfl _ (&fl _ $filst) + filcnt*filstl;
      endda _ $dpyarea + dacnt * dal;
      fileno _ 1;
      UNTIL &fl >= endfl DO
         BEGIN
         IF NOT fl.flnoclos THEN
            IF fl.flexis THEN
               BEGIN
               &da _ $dpyarea;
               used _ FALSE;
               UNTIL &da >= endda DO
                  BEGIN
                  IF filusd(fileno, &da) THEN
                     BEGIN
                     used _ TRUE;
```

```
                         EXIT;
                         END;
                      &da _ &da + dal;
                      END;
                   IF NOT used THEN
                      BEGIN
                      %
                      &da _ $dpyarea;
                      UNTIL &da >= endda DO
                         BEGIN
                         &frr _ da.dalink;
                         endfrr _ &frr + frrhlen + (frrelen*frr.frhlast);
                         FOR &fre _ &frr + frrhlen UP frrelen UNTIL >
                         endfrr DO
                            IF [fre.frring].srhexis AND
                            [fre.frring].srhfileno = fileno THEN
                               [fre.frring].srhfileno _ 0;
                         END;
                      %
                      close(fileno);
                      END;
                   END;
             BUMP fileno;
             &fl _ &fl + filstl;
             END;
          RETURN;
          END.

    (filusd)   PROCEDURE (fileno, da);                                    6B
          %Given a file number and the address of a display area, this
          routine returns TRUE if the file is used in the frozen list or
          csp associated with the display area; otherwise it returns
          FALSE.%
          %----------------%
          LOCAL fl, fz;
          REF fl, fz, da;
          IF da.daempty OR da.dacsp = endfil THEN RETURN(FALSE);
          IF fileno = da.dacsp.stfile THEN RETURN(TRUE);
          &fz _ da.dafrzl;
          WHILE &fz DO
             BEGIN
             IF fz.fzexis AND fz.fzstid.stfile = fileno THEN
                RETURN(TRUE);
             &fz _ fz.fznext;
             END;
          RETURN(FALSE);
          END.

 %.....file name and lock message...%
    (mesfre)PROCEDURE(fl, ptr);                                           7A
          REF fl;
          dismes(2, fl.flastr);
          IF fl.fllock THEN lockmes(ptr.stfile);
          RETURN;
          END.
```

```
%.....generate display.....%
   (recred)                                                            8A
      PROCEDURE;
      IF cdtype = dspno THEN RETURN;%no display necessary%
      %IF NOT namereset THEN dn($""); %%clear name area%
         % what do we do about the name area? %
      IF cdtype = dspallf THEN %recreate all display areas%
         alldsp()
      ELSE seldsp();%selectively recreate display areas%
      RETURN;
      END.
   (alldsp) PROCEDURE;  %recreate display for all display areas%       8B
      %Issues Core-NLS calls to regenerate the display image
      for all of the currently defined text display areas.%
      %-----------------%
      LOCAL da REF, end, y, width;
      LOCAL STRING dtmstg[20];
      IF dacnt NOT IN [1,damax] THEN err($"Fatal display error in
      ALLDSP");
      end _ (&da _ $dpyarea) + dacnt*dal;
      DO IF da.daexis AND NOT da.daseq AND NOT da.dasuppress AND NOT
      da.daauxiliary THEN
         BEGIN
         da.daccnt _ 1;
         dafrmt(&da, 0);
         END
      UNTIL (&da _ &da + dal) = end;
      % Reset global display recreation parameters %
      dpset(dspjpf, endfil, endfil, endfil);
      RETURN;
      END.

   (seldsp) PROCEDURE;  %selective display recreate control%           8C
      %Issues Core-NLS calls to update or reformat the display image
      for each currently defined text display area in which a file that
      was modified is being displayed.  Reformatting is usually needed
      only if current viewspec parameters or the last viewspec change
      make selective updating impossible.%
      %-----------------%
      LOCAL
         frmt,          % call dafrmt flag %
         da,            %temp for walking thru da's%
         f1, f2,        %file numbers of files to be formated%
         end;           %last word address in list of da's%
      LOCAL STRING dtmstg[20];
      REF da;

      %initialization%
         IF dacnt NOT IN [1,damax] THEN
            err($"DACNT out of range; detected in SELDSP");
         %replace all stid's passed only as file indicators, not to be
         used in reformatting%
            f1 _ IF cdstd1 = endfil THEN endfil ELSE cdstd1.stfile;
            f2 _ IF cdstd2 = endfil THEN endfil ELSE cdstd2.stfile;
            CASE cdtype OF
               = dspjpf, = dspyes, =  dspstrc, = dspallf :
                cdstd1 _ cdstd2 _ endfil;
             ENDCASE;
        end _ (&da _ $dpyarea) + dacnt * dal;
     % search through da's %
        DO
           BEGIN %locate candidate for update or reformat%
           IF howformat(&da, f1, f2 : frmt ) THEN %reformat this one%
              BEGIN
              IF frmt THEN %redo whole display%
                 dafrmt(&da, 0)
              ELSE %selectively update display%
                 daupdate (&da);
              (da.dapvs, da.dapvs2) _ (da.davspec, da.davspc2);
              END;
           END
        UNTIL (&da _ &da + dal) = end;
     % Reset global display recreation parameters%
        dpset(dspno, endfil, endfil, endfil);
     RETURN;
     END.

(howformat) PROCEDURE (da, f1, f2);  %test if reformatting should
occur for this display area and ifso, should full reformatting occur
for this display area%                                            8D
     LOCAL mask; %used to mask out certain viewspecs%
     REF da;
     IF da.daexis AND NOT da.daseq AND NOT da.daauxiliary AND NOT
     da.dasuppress AND
     (da.dacsp.stfile = f1 OR
     da.dacsp.stfile = f2 OR
     (da.davspec.vsfrzf AND frzchk(&da, f1, f2))) AND
     (cdtype NOT= dspjpf OR &da = lda())
     THEN %should be reformatted%
        BEGIN %determine how to format it%
        mask _ -1; mask.vslev _ mask.vsrlev _ mask.vslevd _
        mask.vsbrof _ mask.vsplxf _ 0;
           %mask out level fields%
        IF (((da.dapvs) .A mask) # ((da.davspec) .A mask))
           %critical viewspecs have just changed%
        OR cdtype = dspallf
        OR cdtype = dspyes
        OR da.davspec.vscakf
        OR da.davspc2.vsmkrf
        OR da.davspec.vscapf
        OR da.davspec.vsusqf
        OR (da.davspec.vsstnf AND NOT da.davspec.vssidf AND cdtype
        NOT= dsprfmt AND cdtype # dspjpf)
        OR da.daempty
        OR (da.davspec.vsrind AND (da.davspec.vsbrof OR
        da.davspec.vsplxf) AND cdtype NOT= dsprfmt)
        OR (cdtype = dspjpf AND (da.dapstf # da.dacsp.stfile))
        THEN
           RETURN(TRUE, TRUE) %do full reformat%
        ELSE RETURN(TRUE, FALSE); %do partial reformat%
        END
     ELSE RETURN(FALSE, FALSE);
```

```
      END.
   (frzchk) PROCEDURE (da, file1, file2); %check frozen chain for file
   membership%                                                          8E
      %Returns TRUE if there are anyy frozen statements from file1 or
      file2 for display area 'da'; else FALSE%
      LOCAL fz, frzflg;
      REF da, fz;
      IF NOT &fz _ da.dafrzl THEN RETURN  (FALSE); %no chain, this da%
      frzflg _ FALSE; %initial value%
      DO
         BEGIN
         IF fz.fzexis THEN
            IF fz.fzstid.stfile = file1 OR fz.fzstid.stfile = file2
            THEN
               BUMP frzflg
            ELSE NULL
         ELSE err ($"illegal freeze list entry, frzchk");
         END
      UNTIL (&fz _ fz.fznext) = 0;
      RETURN(frzflg);
      END.
%.....build viewspec status string.....%
   (curvsp) PROCEDURE( % convert viewspecs to "human" string %          9A
      vspec,    % address of viewspecs word(s) %
      astrng);  % address of string to append string  %
      LOCAL vspc;
      REF vspec, astrng;
      vspc _ vspec;
      %level%
         *astrng* _ *astrng*, "levels: ";
         IF vspc.vslev = 63 THEN *astrng* _ *astrng*, "ALL"
         ELSE *astrng* _ *astrng*, STRING(vspc.vslev);
      %truncation%
         *astrng* _ *astrng*, ", lines: ";
         IF vspc.vstrnc = 63 THEN *astrng* _ *astrng*, "ALL"
         ELSE *astrng* _ *astrng*, STRING(vspc.vstrnc);
         *astrng* _ *astrng*, ", ";
      %branch only stuff%
         *astrng* _ *astrng*,
            IF vspc.vsbrof THEN 'g
            ELSE IF vspc.vsplxf THEN 'l
               ELSE 'h;
      %content analysis%
         *astrng* _ *astrng*,
            IF vspc.vscapf THEN 'i
            ELSE IF vspc.vscakf THEN 'k
               ELSE 'j;
      %statement numbers%
         *astrng* _ *astrng*,
            IF vspc.vsstnf THEN 'm ELSE 'n;
      %frozen stats%
         *astrng* _ *astrng*,
            IF vspc.vsfrzf THEN 'o ELSE 'p;
      %formatter on/off%
         *astrng* _ *astrng*,
            IF vspc.vsdaft THEN 'u ELSE 'v;
      %blank lines%
         *astrng* _ *astrng*,
            IF vspc.vsblkf THEN 'y ELSE 'z;
      %indenting%
         *astrng* _ *astrng*,
            IF vspc.vsrind THEN 'Q ELSE IF vspc.vsindf THEN 'A ELSE 'B;
      %names%
         *astrng* _ *astrng*,
            IF vspc.vsnamf THEN 'C ELSE 'D;
      %Pagination%
         *astrng* _ *astrng*,
            IF vspc.vspagf THEN 'E ELSE 'F;
      %stat nums left/right%
         *astrng* _ *astrng*,
            IF vspc.vsstnr THEN 'G ELSE 'H;
      %stat nums or SID's%
         *astrng* _ *astrng*,
            IF vspc.vssidf THEN 'I ELSE 'J;
      %signature%
         *astrng* _ *astrng*,
            IF vspc.vsidtf THEN 'K ELSE 'L;
      %user sequence generator%
         *astrng* _ *astrng*,
            IF vspc.vsusgf THEN 'O ELSE 'P;
      RETURN END.
  %.....freeze statement support.....%
    (xrelsta)                                                          10A
      %This routine searches the chain of frozen statements.
      If the stid passed it is in the frozen list for the display
      area passed it, the routine removes it from the list,
      squeezes the frozen list, and adds the deleted element to
      the fozen element free list.%
      %------------------%
      PROCEDURE(dpa, stid);
      LOCAL frzprev, frzelm;
      REF dpa, frzprev, frzelm;
      IF &frzprev _ &frzelm _ dpa.dafrzl THEN LOOP
         BEGIN
         IF frzelm.fzstid = stid THEN
            BEGIN
            IF &frzelm = &frzprev THEN %first item in list%
               dpa.dafrzl _ frzelm.fznext
            ELSE frzprev.fznext _ frzelm.fznext;
            frzelm.fzexis _ FALSE;
            frzelm.fznext _ fzfree := &frzelm;
            EXIT;
            END;
         &frzprev _ &frzelm;
         IF frzelm.fznext THEN &frzelm _ frzelm.fznext
         ELSE EXIT;
         END;
      RETURN;
      END.
    (xrelallsta)                                                       10B
      %Given the address of a display area, this routine will
```

```
   free all of the frozen elements associated with the area.%
   %-----------------%
   PROCEDURE(dpa);
   LOCAL frzelm;
   REF dpa, frzelm;
   IF &frzelm _ dpa.dafrzl THEN
      BEGIN
      LOOP
         BEGIN
         frzelm.fzexis _ FALSE;
         IF NOT frzelm.fznext THEN EXIT
         ELSE &frzelm _ frzelm.fznext;
         END;
      frzelm.fznext := fzfree := dpa.dafrzl := 0;
      END;
   RETURN;
   END.

(Xfresta)                                                        10C
   %Given the addrss of a display are, an stid, and two
   viewspec words, this routine will create a frozen element
   for the stid passed it, in the display area passed.  If
   the stid is already on the frozen lit for this area, the
   new vspec words will replace the old ones for that element.%
   %-----------------%
   PROCEDURE(dpa, stid, vspec1, vspec2);
   LOCAL frzelm;
   REF dpa, frzelm;
   IF &frzelm _ dpa.dafrzl THEN
      BEGIN
      LOOP
         BEGIN
         IF frzelm.fzstid = stid THEN
            BEGIN
            frzelm.fzvspec _ vspec1;
            frzelm.fzvspc2 _ vspec2;
            RETURN;
            END;
         IF frzelm.fznext THEN &frzelm _ frzelm.fznext
         ELSE EXIT;
         END;
      IF NOT fzfree THEN err(5);
      &frzelm _ frzelm.fznext _ fzfree := [fzfree].fznext;
      frzelm.fznext _ 0;
      END
   ELSE
      BEGIN
      IF NOT fzfree THEN err(5);
      dpa.dafrzl _ &frzelm _ fzfree := [fzfree].fznext;
      frzelm.fznext _ 0;
      END;
   frzelm.fzstid _ stid;
   frzelm.fzvspec _ vspec1;
   frzelm.fzvspc2 _ vspec2;
   frzelm.fzexis _ TRUE;
   RETURN;
```

```
      END.

%.....screen spliting Support Routines.....%
   (wbreak)   % CL:LB ;  Break window %
   PROCEDURE (atbug REF LIST, centered, direction, dspinbug REF LIST %
   => &daold, &danew  %);                                          11A
      % Procedure description
         FUNCTION
            Break a window into 2 windows.  The window is split in half
            if centered is TRUE or through point atbug if centered is
            FALSE. The data in the original window is displayed in the
            window that contains the point dspinbug.  Two new windows
            are created and the original window is deleted unless it is
            the primary window.  If the original window is the primary
            window, the two new windows are created to overlap the
            original window (in the FE) and the BE data structures that
            originally refered to the primary window now refer to one
            of the new windows.
         ARGUMENTS
            atbug:  point (coordinate selection) at which to break
            window
            centered:  TRUE to break window into 2 equal parts.
            direction:  vertical to break window vertically; horizontal
            to break window horizontally
            dspinbug:  point (coordinate selection) at which to display
            orginal data
         RESULTS
            daold:  address of da for window containing data in
            original window
            danew:  address of da for other new  window
         NON-STANDARD CONTROL
            ABORT if invalid break
         GLOBALS
            Set cwndow to new window-id for original da
            dspcmd:  set and null
         %
      % Declarations %
         LOCAL wid, waold REF, wanew REF _ 0, daold REF, danew REF _ 0,
         temp;
         LOCAL LIST retlist[2];
         REF dspcmd, prwndw;
      % Invoke catchphrase %
         INVOKE (catwbreak);
      % Null out commands list %
         IF dspcmd.L THEN #dspcmd# _ ;
      % Check that displaying point is in window being split %
         IF (wid _ ELEM #atbug#[wndw]) NOT= ELEM #dspinbug#[wndw] THEN
            ABORT(erdwindow, $"Displaying point not in window");
      % Set up wa's and da's %
         &waold _ findwa(wid);
         IF NOT (&daold _ dsparea(wid)) THEN
            ABORT(erdprogram, $"Error in display area - wbreak");
         &wanew _ newwa();
         &danew _ newda();
         copywa(&waold, &wanew);
         copyda(&daold, &danew);
      % Set new da to empty %
         danew.daempty _ TRUE;
         danew.dacsp _ endfil;
      % link new wa to new da %
         wanew.widdarea _ &danew;
   % Check if first screen split, i.e. "old" window is the primary
   window %
      IF &waold = &prwndw THEN
         BEGIN  % first screen split %
         clearda(&daold, &dspcmd);  %clear primary window in FE%
         %save primary window data%
            &prwndw _ newwa();
            copywa(&waold, &prwndw);
         prwndw.widdarea _ 0;  %no da associated with it any more%
         BUMP DOWN waold.wipriority;  % overlay primary window %
         BUMP DOWN wanew.wipriority;  % overlay primary window %
         waold.widowner _ wid;  %primary window is owning window%
         wanew.widowner _ wid;  %primary window is owning window%
         END
      ELSE blddw(&waold, &dspcmd);  %delete window in FE%
   % Calculate new boundaries and place in wa's and da's %
      wbbnd(&waold, &wanew, &atbug, centered, direction, &dspinbug);
   % Send commands to FE %
      % Build commands to create new windows %
         bldcw(&waold, &dspcmd);
         bldcw(&wanew, &dspcmd);
      % Send commands to FE and store new window ids's %
         prcmds(FALSE, &dspcmd, FALSE, $retlist);
         cwndow _ waold.widindex _ ELEM #retlist#[1];
         daold.dawid _ waold.widindex;
         wanew.widindex _ ELEM #retlist#[2];
         danew.dawid _ wanew.widindex;
   % Return %
      #dspcmd# _ ;
      #retlist# _ ;
      DROP (catwbreak);
      RETURN( &daold, &danew );
   % Catchphrases %
      (catwbreak)  CATCHPHRASE();                                   11A11A
         BEGIN
         DISABLE (catwbreak);
         CASE SIGNALTYPE OF
            = notetype:
               CASE SIGNAL OF
                  = return, = unwind :
                     BEGIN  %null lists%
                     #dspcmd# _ ;
                     #retlist# _ ;
                     END;
               ENDCASE CONTINUE;
            = aborttype:
               BEGIN
               IF &wanew THEN
                  BEGIN
                  temp _ &wanew := 0;
                  delwa(temp);
```

```
                    END;
                 IF &danew THEN
                    BEGIN
                    temp _ &danew := 0;
                    delda(temp);
                    END;
                 END;
            ENDCASE CONTINUE;
         END;
      END.

(wbbnd)   % CL:LP ;  calculate boundaries for break window command %
PROCEDURE (waold REF, wanew REF, atbug REF LIST, centered,
direction, dspinbug REF LIST);                                   11B
   % Procedure description
      FUNCTION
         Calculate boundaries for new windows created by a break
         window command.  Place new boundaries in wa's and da's
         according to dspinbug, i.e. set boundaries of old wa and da
         so that it includes the point dspinbug (where data is to be
         re-displayed).
      ARGUMENTS
         waold:  original window window area
         wanew:  new window window area
         atbug:  point in window to split
         centered:  TRUE to split window into 2 equal parts.  FALSE
         to split at point atbug
         direction:  vertical for vertical break;  horizontal for
         horizontal break
         dspinbug:  point in window in which to re-display data
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         ABORT if invalid split -- window is too small to split
      GLOBALS
         none
      %
   % Declarations %
      LOCAL top, bottom, left, right, wsize, newcoord, atxc, atyc,
      dspxc, dspyc;
   % Set local boundary variables %
      left _ waold.wuplx;
      right _ waold.wlrx;
      top _ waold.wuply;
      bottom _ waold.wlry;
   % Calculate boundaries for 2 new windows %
      % resolve bugs to owning window coordinates %
         atxc _ resown(&atbug: atyc);
         dspxc _ resown(&dspinbug: dspyc);
      CASE direction OF
         =vertical:
            BEGIN
            wsize _ right - left + 1;
            IF wsize < minww*2 THEN
               ABORT (erdwindow, $"Window too small");
            IF centered THEN newcoord _ (wsize / 2) + left - 1
```

```
          ELSE newcoord _ atxc;
          IF dspxc <= newcoord THEN
             BEGIN  %display point is in left half%
             IF NOT centered THEN
                BEGIN
                % change break point if either window would be
                less than minimum size %
                   newcoord _ MAX(newcoord, left - 1 + minww);
                   newcoord _ MIN(newcoord, right - minww);
                END;
             waold.wlrx _ newcoord;
             wanew.wuplx _ newcoord + 1;
             END
          ELSE
             BEGIN  %display point is in right half%
             IF centered THEN
                BEGIN
                waold.wuplx _ newcoord + 1;
                wanew.wlrx _ newcoord;
                END
             ELSE
                BEGIN
                % change break point if either window would be
                less than minimum size %
                   newcoord _ MAX(newcoord, left + minww);
                   newcoord _ MIN(newcoord, right + 1 - minww);
                waold.wuplx _ newcoord;
                wanew.wlrx _ newcoord - 1;
                END;
             END;
          END;
       =horizontal:
          BEGIN
          wsize _ top - bottom + 1;
          IF wsize < minwh*2 THEN
             ABORT (erdwindow, $"Window too small");
          IF centered THEN newcoord _ (wsize / 2) + bottom - 1
          ELSE newcoord _ atyc;
          IF dspyc <= newcoord THEN
             BEGIN  % display point is in bottom window %
             IF NOT centered THEN
                BEGIN
                % change break point if either window would be
                less than minimum size %
                   newcoord _ MAX(newcoord, bottom - 1 + minwh);
                   newcoord _ MIN(newcoord, top - minwh);
                END;
             waold.wuply _ newcoord;
             wanew.wlry _ newcoord + 1;
             END
          ELSE
             BEGIN  % display point is in top window %
             IF centered THEN
                BEGIN
                waold.wlry _ newcoord + 1;
                wanew.wuply _ newcoord;
```

```
                  END
               ELSE
                  BEGIN
                  % change break point if either window would be
                  less than minimum size %
                     newcoord _ MAX(newcoord, bottom + minwh);
                     newcoord _ MIN(newcoord, top + 1 - minwh);
                  waold.wlry _ newcoord;
                  wanew.wuply _ newcoord - 1;
                  END;
               END;
            END;
         ENDCASE ABORT (erdprogram, $"Invalid direction - wbbnd");
   % Set margins and related fields in display areas %
      setdabnd(&waold, waold.widdarea);
      setdabnd(&wanew, wanew.widdarea);
   % Return %
      RETURN;
   END.

(setdabnd)   % CL:LB ;  set display area margins from window area %
PROCEDURE (wa REF, da REF);                                          11C
   % Procedure description
      FUNCTION
         Set display area margins.  In display area (0,0) is upper
         left corner,  (0,window height) is lower left corner,
         (window width,0) is upper right corner,  (window
         width,window height) is lower right corner.  This is
         different from the window as viewed in the front-end where
         (0,0) designates the lower left corner of the window.
         Also, the boundaries in the window area are with respect to
         the owning window and, as stated above, the margins in the
         display area are with respect to the window itself.
      ARGUMENTS
         wa:  window area address
         da:  display area address
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         ABORT if display area field in wa is empty
      GLOBALS
         none
      %
   % Declarations %
   % Set margins %
      da.daleft _ 0;
      da.daright _ wa.wlrx - wa.wuplx;
      da.dabottom _ wa.wuply - wa.wlry;
      da.datop _ 0;
   % Set margin dependent fields %
      da.damrow _ IF nlmode NOT= typewriter THEN
         ((da.dabottom)/vinc)*vinc ELSE linmax %lines to bottom
         margin%;
      %IF nldevice = devlproc THEN da.damrow _ da.damrow - vinc;%
      IF nlmode= typewriter THEN
         da.damcol _ colmax * hinc
      ELSE
         BEGIN
         IF udpcolmax AND da.daright/hinc  > udpcolmax THEN
            da.damcol _ (udpcolmax - 1) * hinc
               %user specifies right margin starting at column 1 but
               formatting of display starts column count at 0%
         ELSE da.damcol _ (da.daright/hinc) * hinc;
         END;
   % Return %
      RETURN;
   END.

(wappend)   % CL:LB ;  append (merge) two windows %
PROCEDURE (frombug REF LIST, tobug REF LIST % => &dato %);              11D
   % Procedure description
      FUNCTION
         Delete the window at frombug expanding the window at tobug
         to include the portion of the screen occupied by the
         deleted window.
      ARGUMENTS
         frombug:  point in window to be deleted
         tobug:  point in window to be expanded
      RESULTS
         dato:  da for expanded window
      NON-STANDARD CONTROL
         ABORT if the 2 windows do not have the same owning window
         ABORT if the 2 windows do not have one entire boundary in
         common
         ABORT if da not defined for either window
      GLOBALS
         Set cwndow to expanded window
         dspcmd:  set and null
      %
   % Declarations %
      LOCAL btype, wafrom REF, dafrom REF, wato REF, dato REF;
      LOCAL LIST retlist[1];
      REF dspcmd;
   % Invoke catchphrase %
      INVOKE (catwappend);
   % Null out commands list %
      IF dspcmd.L THEN #dspcmd# _ ;
   % Check for same owning window and adjacent boundaries %
      &wafrom _ findwa(ELEM #frombug#[wndw]);
      &wato _ findwa(ELEM #tobug#[wndw]);
      IF NOT (&dato _ wato.widdarea) OR NOT (&dafrom _
      wafrom.widdarea) THEN
         ABORT(erdprogram, $"Error in display area - wappend");
      IF wafrom.widowner NOT= wato.widowner OR NOT combnd(&wafrom,
      &wato: btype) THEN
         ABORT(erdwindow, $"Only rectangular windows are valid");
   % Expand "to" window %
      CASE btype OF
         = bbound, =tbound:  %adjacent horizontal boundary%
            BEGIN  % height of window increases %
            wato.wuply _ MAX(wato.wuply, wafrom.wuply);
            wato.wlry _ MIN(wato.wlry, wafrom.wlry);
            END
      ENDCASE  %adjacent vertical boundary%
         BEGIN  % width of window increases %
         wato.wuplx _ MIN(wato.wuplx, wafrom.wuplx);
         wato.wlrx _ MAX(wato.wlrx, wafrom.wlrx);
         END;
   % Adjust margins and related fields in expanded display area %
      setdabnd(&wato, &dato);
   % Send dspcmd to FE to both windows and create one large window %
      % delete both windows in FE %
         blddw(&wato, &dspcmd);
         blddw(&wafrom, &dspcmd);
      % create new expanded window %
         bldcw(&wato, &dspcmd);
      % send command to FE and store new wid %
         prcmds(FALSE,&dspcmd, FALSE, $retlist);
         cwndow _ wato.widindex _ ELEM #retlist#[1];
         dato.dawid _ wato.widindex;
   % Delete wa and da for deleted window %
      clrall(&dafrom,TRUE);  % free string table space %
      delda(&dafrom);
      delwa(&wafrom);
   % Null list %
      #dspcmd# _ ;
   % Return %
      DROP(catwappend);
      RETURN(&dato);
   % Catchphrase %
      (catwappend)  CATCHPHRASE();                                 11D12A
         BEGIN
         DISABLE (catwappend);
         CASE SIGNALTYPE OF
            = notetype:
               CASE SIGNAL OF
                  = return, = unwind :
                     BEGIN  %null lists%
                     #dspcmd# _ ;
                     END;
               ENDCASE CONTINUE;
         ENDCASE CONTINUE;
         END;
   END.

(movbndry)   % CL:LB ;  move window boundary %
PROCEDURE (frombug REF, tobug REF);                                   11E
   % Procedure description
      FUNCTION
         Move a window boundary, expanding "from" window and
         contracting the adjacent window ("to" window).  The
         boundary of the "from" window that is adjacent to a
         boundary in the "to" window will be movd to pass through
         the point tobug.
         This routine uses dafrmt to update the 2 affected windows.
      ARGUMENTS
         frombug:  point in window to be expanded
         tobug:  point to which window is to be expanded
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         ABORT if the 2 windows do not have the same owning window
         ABORT if boundary is common to more than 2 windows
         ABORT either window would be too small
      GLOBALS
         Set cwndow to the new window id of the "from" window
         dspcmd:  set and null
      %
   % Declarations %
      LOCAL xc, yc, wafrom REF, dafrom REF, waneighbor REF,
      daneighbor REF, type;
      LOCAL LIST retlist[2];
      REF dspcmd;
   % Invoke catchphrase %
      INVOKE (catmovbnd);
   % Null out commands list %
      IF dspcmd.L THEN #dspcmd# _ ;
   % Get wa's and da's %
      &wafrom _ findwa(ELEM #frombug#[wndw]);
      &waneighbor _ findwa(ELEM #tobug#[wndw]);
      IF NOT ((&dafrom _ wafrom.widdarea) AND (&daneighbor _
      waneighbor.widdarea)) THEN
         ABORT(erdprogram,$"Error in display area - movbndry");
   % Check that that 2 windows are adjacent and have same owning
   window %
      IF NOT ((wafrom.widowner = waneighbor.widowner) AND
      combnd(&wafrom, &waneighbor: type)) THEN
         ABORT(erdwindow, $"Invalid Expand");
   % Move the boundary (affects 2 adjacent windows) %
      % Resolve tobug to owning window coordinates %
         xc _ resown(&tobug: yc);
      CASE type OF
         =tbound:  %top boundary of window, bottom boundary of
         neighbor%
            BEGIN
            IF yc > waneighbor.wuply - minwh THEN
               ABORT (erdwindow, $"Window too small");
            wafrom.wuply _ yc;
            waneighbor.wlry _ yc + 1;
            END;
         =bbound:  %bottom boundary of window, top boundary of
         neighbor%
            BEGIN
            IF yc < waneighbor.wlry + minwh THEN
               ABORT (erdwindow, $"Window too small");
            wafrom.wlry _ yc;
            waneighbor.wuply _ yc - 1;
            END;
         =lbound:  %left boundary of window, right boundary of
         neighbor%
            BEGIN
            IF xc < waneighbor.wuplx + minww THEN
               ABORT (erdwindow, $"Window too small");
            wafrom.wuplx _ xc;
            waneighbor.wlrx _ xc - 1;
            END;
         =rbound:  %right boundary of window, left boundary of
         neighbor%
            BEGIN
            IF xc > waneighbor.wlrx - minww THEN
               ABORT (erdwindow, $"Window too small");
            wafrom.wlrx _ xc;
            waneighbor.wuplx _ xc + 1;
            END;
      ENDCASE;
   % Adjust margins and related fields in da's %
      setdabnd(&wafrom, &dafrom);
      setdabnd(&waneighbor, &daneighbor);
   % Send dspcmd to FE to delete both windows and create new ones %
      % delete both windows in FE %
         blddw(&wafrom, &dspcmd);
         blddw(&waneighbor, &dspcmd);
      % create new expanded window %
         bldcw(&wafrom, &dspcmd);
         bldcw(&waneighbor, &dspcmd);
      % send command to FE and store new wid's %
         prcmds(FALSE,&dspcmd, FALSE, $retlist);
         cWndow _ wafrom.widindex _ ELEM #retlist#[1];
         dafrom.dawid _ wafrom.widindex;
         waneighbor.widindex _ ELEM #retlist#[2];
         daneighbor.dawid _ waneighbor.widindex;
   % Update screen for both windows %
      dafrmt(&dafrom, 0);
      dafrmt(&daneighbor, 0);
   % Return %
      DROP (catmovbnd);
      RETURN;
   (catmovbnd)  CATCHPHRASE();                                11E12
      BEGIN
      DISABLE (catmovbnd);
      CASE SIGNALTYPE OF
         = notetype:
            CASE SIGNAL OF
               = return, = unwind :
                  BEGIN  %null lists%
                  #dspcmd# _ ;
                  END;
            ENDCASE CONTINUE;
      ENDCASE CONTINUE;
      END;
   END.

(combnd)   % CL:LB ;  see if 2 windows have adjacent boundaries %
PROCEDURE (wa1 REF, wa2 REF % => adjacent, btype %);          11F
   % Procedure description
      FUNCTION
         Given 2 window areas, compare diagonal coordinates to see
         if windows have vertical or horizontal boundaries that
         differ by 1 increment
      ARGUMENTS
```

```
         wa1:  window area address
         wa2:  window area address
      RESULTS
         adjacent:  TRUE if windows have adjacent boundaries
         btype:  which boundary of wa1 is adjacent to wa2
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % Declarations %
      LOCAL adjacent _ FALSE, btype;
   % Compare diagonal coordinates %
      IF wa1.wuply = wa2.wuply AND wa1.wlry = wa2.wlry THEN
         BEGIN  %same vertical position on screen%
         IF (wa2.wuplx - wa1.wlrx) = 1 THEN
            BEGIN
            adjacent _ TRUE;
            btype _ rbound;  %left boundary of wa1%
            END
         ELSE
            IF (wa1.wuplx - wa2.wlrx) = 1 THEN
               BEGIN
               adjacent _ TRUE;
               btype _ lbound;  %right boundary of wa1%
               END;
         END
      ELSE
         IF wa1.wuplx = wa2.wuplx AND wa1.wlrx = wa2.wlrx THEN
            BEGIN  %same horizontal position on screen%
            IF (wa1.wlry - wa2.wuply) = 1 THEN
               BEGIN
               adjacent _ TRUE;
               btype _ bbound;  %bottom boundary of wa1%
               END
            ELSE
               IF (wa2.wlry - wa1.wuply) = 1 THEN
                  BEGIN
                  adjacent _ TRUE;
                  btype _ tbound;  %top boundary of wa1%
                  END
            END;
   % Return %
      RETURN(adjacent, btype);
   END.

(delda)   % CL:LB ;  delete display area %
PROCEDURE (da REF);                                                   11G
   % Procedure description
      FUNCTION
         Return da to da pool.  Return allocated storage for string
         table, return ring and frozen statement chain.
      ARGUMENTS
         da:  display area address
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % Declarations %
      LOCAL end, lfrozen REF;
   %put frzlst entries on free list%
      IF &lfrozen _ da.dafrzl THEN
         BEGIN
         WHILE &lfrozen DO
            BEGIN
            lfrozen.fzexis _ FALSE;
            &lfrozen _ lfrozen.fznext;
            END;
         lfrozen.fznext := fzfree := da.dafrzl := 0;
         END;
   %release link stack%
      treefrring(da.dalink := 0);
   %deallocate core associated with LSRT%
      IF da.dastrt THEN dlstbl(da.dastrt, TRUE);
   %clear out da%
      end _ &da + dal;
      DO da _ 0 UNTIL (&da _ &da + 1) = end;
   % Return %
      RETURN;
   END.

(delwa)   % CL:LB ;  delete window area %
PROCEDURE (wa REF);                                                    11H
   % Procedure description
      FUNCTION
         Return wa to wa pool.  Associated data structures (da,
         string table, etc., are not affected).
      ARGUMENTS
         wa:  window area address
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % Declarations %
      LOCAL end;
   %clear out wa%
      end _ &wa + wal;
      DO wa _ 0 UNTIL (&wa _ &wa + 1) = end;
   % Return %
      RETURN;
   END.

(resown)   % CL:LB ;  resolve bug to owning window coordinates %
PROCEDURE (bug REF % => xc, yc %);                                     11I
   % Procedure description
      FUNCTION
```

```
         Resolve a bug (point selection) to a coordinate pair with
         respect to the owning window of the window at the bug
      ARGUMENTS
         bug:  point selection from fe
      RESULTS
         xc:  x-coordinate with respect to owning window
         yc:  y-coordinate with respect to owning window
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % Declarations %
      LOCAL wa REF;
   % Add coordinates of window origin to get owninw window
   coordinates %
      &wa _ findwa(ELEM #bug#[wndw]);  % get wa %
   % Return %
      RETURN(wa.wuplx + ELEM #bug#[xcoord], wa.wlry + ELEM
      #bug#[ycoord]);
   END.

%.....overlapping window Support Routines.....%
  (wdelete) PROCEDURE;  %delete window%                          12A
     ABORT(notyet, $"Not implemented.");
     END.
  (pushda) % save current display area and overlay a new one %
  PROCEDURE (priority % => da, wa %);                            12B
     % Procedure description
        FUNCTION
           For subsystems or commands that want to temporarily create
           a window on top of existing windows's.  Should be used by
           Help, the calculator "Display" command, and Fill.
        ARGUMENTS
           priority -- an integer.  1 is the highest priority.  It is
           unclear what will happen if you overlay two windows with
           priority 1.  There is a primitive in the back end for
           changing the priorities of windows.
        RESULTS
           returns the addresss of a display area and a window area.
           These are accepted by popda to delete the overlay.
        NON-STANDARD CONTROL
           none
        GLOBALS
           dspcmd
           endfil
           dspno
           widowner
           wipriority
           widdarea
           dpyarea
           dacnt
           dal
           calcaux
           cda
           daauxiliary
         nlmode
         fulldisplay
      %
   % declarations %
      LOCAL end, da REF, wa REF;
      LOCAL LIST retlist[2];
      REF prwndw, qda, qwa, dspcmd;
   IF nlmode = fulldisplay THEN
      BEGIN
      % Save away auxiliary bit for new display area. %
         calcaux _ cda.daauxiliary ;
      % set bit in da's so that they are ignored. %
         end _ (&da _ $dpyarea) + dacnt*dal;
         DO da.daauxiliary _ TRUE
         UNTIL (&da _ &da + dal) >= end;
      END;
   &da _ newda();
   intdafl(&da);
   IF nlmode = fulldisplay THEN
      BEGIN
      &wa _ newwa();
      copywa(&prwndw, &wa);
      wa.widdarea _ &da;
      wa.wipriority _ priority;
      wa.widowner _ prwndw.widindex;
      #dspcmd# _ ;  %null out commands list%
      bldcw(&wa, &dspcmd);
      prcmds(FALSE, &dspcmd, FALSE, $retlist);
      #dspcmd# _ ;  %null out commands list%
      da.dawid _ wa.widindex _ ELEM #retlist#[1];
      dpset(dspno, endfil, endfil, endfil);
      END;
   RETURN (&da, $wa);
   END.

(popda)  % delete given da and wa.  In dnls, restore old da %
PROCEDURE (da REF, wa REF);                                          12C
   % Procedure description
      FUNCTION
         For deleting a window created using pushda and restoring
         what was under it.
      ARGUMENTS
         da -- address of display area returned from pushda
         wa -- address of window area returned from pushda
      RESULTS
         none
      NON-STANDARD CONTROL
         none
      GLOBALS
         dspcmd
         endfil
         dspno
         dpyarea
         dacnt
         dal
         calcaux
```

```
            cda
            daauxiliary
            nlmode
            fulldisplay
         %
      % declarations %
         LOCAL locda REF, end;
         LOCAL LIST retlist[2];
         LOCAL STRING vspstr[16];
         REF vswndw, dspcmd;
      delda(&da:=0);
      IF nlmode = fulldisplay THEN
         BEGIN
         % Get rid of data structures associated with window in FE and
         BE. %
            #dspcmd# _ ;
            IF &wa THEN blddw(&wa, &dspcmd);
            prcmds(FALSE, &dspcmd, FALSE, $retlist);
            #dspcmd# _ ;
            delwa(&wa := 0);
         %  set bit in da's so that they are not ignored.  %
            end _ (&locda _ $dpyarea) + dacnt*dal;
            DO locda.daauxiliary _ FALSE
            UNTIL (&locda _ &locda + dal) >= end;
         %  Restore auxiliary display area.  %
            cda.daauxiliary _ calcaux ;
         dpset(dspno, endfil, endfil, endfil);
         END;
      NULL-LISTS;
      RETURN;
      END.

%.....get input from the user -- askuser.....%
   (askuser) % ask the user and get input %
   PROCEDURE (noisewords REF, seltype, enttype, reflist REF LIST %=>
   TRUE/FALSE%);                                                  13A
      % Procedure description
         FUNCTION
            given noise words, a selection type, an entity type (use
            typchoice if the user is to specify the kind of entity) and
            a place for a list, gets input from the user.  If entities
            are strings, reflist will contain actual string.  For
            structure, reflist will contain two text pointers.  Returns
            true.  Returns entity type in case it was user specified.
            Returns false if user hits CD.
         ARGUMENTS
            address of a string containing noisewords
            selection type: lseltyp, sseltyp, [dseltyp, answtyp, etc.
            not implemented]
            entity type:  cwstatement, cwbranch, cwgroup, cwplex,
            cwcharacter, cwword, cwtext, cwvisible, cwinvisible,
            cwlink, cwnumber or
               cwstructure, if the user is to specify the structure
               entity
               cwstring, if the user is to specify the string entity
         RESULTS
```

```
            enttype
         NON-STANDARD CONTROL
            none
         GLOBALS
            entity types and selection types
         %
      % declarations %                                                    13A2
         LOCAL LIST retlist[2], loclist[2];
         LOCAL ptr1 REF, ptr2 REF;
         LOCAL STRING locstring[1999];
      % set up param list and call helpfe %                               13A3
         #loclist#[1] _ seltype;
         #loclist#[2] _ enttype;
         IF NOT helpfe(htyptg, &noisewords, $loclist, $retlist) THEN
            BEGIN
            #loclist# _ ;
            RETURN(FALSE);
            END;
      % get rid of extra list level %
         #reflist# _ COPY #[#retlist#[1]]#;
      % check if result needs conversion %
         IF reflist.L >= 2 % NLS type selection %
         AND isstctype( getvtyp(DESCR #reflist#[tppair]) ) THEN
            BEGIN
            % update entity type to the one actually selected %
               enttype _ ELEM #reflist#[1];
            CASE TRUE OF
               = isstcentity( enttype ):
                  % doesn't need conversion -- get rid of token value %
                                                                    13A5B3A1
                     #reflist#[1] _ ;
               ENDCASE % needs conversion %
                  BEGIN % get actual string %
                  &ptr1 _ #reflist#[tppair];
                  &ptr2 _ &ptr1+d2sel;
                  *locstring* _ ptr1 ptr2;
                  #reflist# _ *locstring*;
                  END;
            END;
      #retlist# _ ;
      #loclist# _ ;
      % return %
         RETURN(TRUE, enttype) ;
      END.
%.....record session support.....%
   (xrecplasup) %record or playback from file%                           14A
      PROCEDURE(record, f1, f2, rhostn);
      LOCAL jfn, gtjflg, opnflg;
      LOCAL STRING fname[80];
      REF f1, f2;
      CASE rhostn OF
         = lhostn: NULL;
         ENDCASE err($"Remote File Manipulations Not Implemented Yet");
      IF record THEN %recording input for later playback%
         BEGIN
         opnflg _ write;
         gtjflg _ gtjoof;
         END
      ELSE
         BEGIN
         opnflg _ read;
         gtjflg _ gtjoif;
         END;
      *fname* _ f1 f2;
      IF NOT jfn _ lgetjfn(0, $fname, $ctlext, gtjflg, $lit) THEN
         err($lit);
      IF NOT sysopen(jfn, opnflg, chrtyp, $lit) THEN
         BEGIN
         reljfn(jfn);
         err($lit);
         END;
      IF record THEN
         BEGIN
         inpwatchjfn _ jfn;
         %for bin/bout zero byte anomoly%
            R1 _ jfn;
            R2 _ 1;
            !JSYS bout;
         END
      ELSE
         BEGIN
         inpjfn _ jfn;
         R1 _ nlssbn _ <AUXCOD, getsbn>($"DNLCTL");
         %for bin/bout zero byte anomoly%
            R1 _ jfn;
            !JSYS bin;
         END;
      &rawchr _ $cntrlgetchar;
         %change lowest level input routine to be one which knows about
         control stuff%
      RETURN;
      END.

%.....auxiliary input stuff....%
   (auxstartup) %***% PROCEDURE( %setup to do input from auxilary
   source%                                                            15A
      ptr1, %pointer to TP for start of auxilary text%
      ptr2); %pointer to TP for end of auxilary text%
      LOCAL f1;
      REF ptr1, ptr2, f1;
       % change dispatch for "rawchr" to auxilary routine %
         auxsav _ &rawchr;  %save current character dispatch%
         &rawchr _ $auxchr; %and subsitute mine(1st time guy)%
      % setup work area and save text pointers %
         auxwrk _ auxtp1 _ ptr1;
         auxwrk[1] _ auxtp1[1] _ ptr1[1];
         auxtp2 _ ptr2;
         auxtp2[1] _ ptr2[1];
      % nail down file if not a-string %
         IF NOT auxwrk.stastr THEN
            BEGIN
            &f1 _ flntadr(auxwrk.stfile);
```

```
          fl.flnoclos _ TRUE;
          END;
      % pass over statement names and setup work area %
         IF NOT auxwrk.stastr THEN auxwrk[1] _ fchtxt(auxwrk);
         fechc1(forward, $auxwrk);
      % jam recognition mode to be "demand" %
         %  should be a call to the FE to change recognition mode
         auxmod _ recogmode; %%1st save current one %%
         auxmd2 _ recog2mode;
         recogmode _ mdemand;
         recog2mode _ mdemand;
         %
      % force to recognize upper/lower case %
         CASE nldevice OF
            = dev33, = dev35, =offline:  initch(nettty);
            ENDCASE;
      RETURN;
      END.
   (auxchr) PROCEDURE; % *** % %get characters for executable text%
                                                                      15B
      LOCAL char;
      LOOP
         IF (NOT inptrf) AND (auxwrk # auxtp2 OR POS auxwrk < auxtp2)
         THEN %characters left to get%
            BEGIN
            INVOKE (sigaux);
            CASE (char _ READC($auxwrk)) OF      %get the next char%
               = ENDCHR:
                  IF NOT basestateflag THEN
                     BEGIN
                     % continue with normal input if end of statement
                     and in parse or execution%
                     auxinterminate(); %go terminate auxilary input%
                     DROP (sigaux);
                     RETURN(rawchr());
                     END
                  ELSE
                     BEGIN
                     auxwrk _ getnxt(auxwrk);
                     auxwrk[1] _ fchtxt(auxwrk);
                     fechc1(forward, $auxwrk);
                     REPEAT LOOP;
                     END
               ENDCASE
                  BEGIN
                  IF echofg THEN typech(char);
                  DROP (sigaux);
                  RETURN(char);
                  END;
            END
         ELSE
            BEGIN
            auxinterminate(); %go terminate auxilary input%
            DROP (sigaux);
            IF inptrf THEN
               BEGIN
            dismes(2, $"User Terminated Input");
            RETURN(CD);
            END
          ELSE RETURN(rawchr());
          END;


    (sigaux) CATCHPHRASE;                                                   15B3
        BEGIN
        DISABLE (sigaux);
        auxinterminate();
        CONTINUE;
        END;
    END.
  (auxinterminate) PROCEDURE; %terminate auxilary input%                    15C
    LOCAL fl;
    REF fl;
    % reset recognition mode and char input routine %
        &rawchr _ auxsav;        %reset where to get characters%
        % should be done thru the FE
        recogmode _ auxmod;     %%reset recognition mode%%
        recog2mode _ auxmd2;
        %
    % free up file if not a-string %
        IF NOT auxwrk.stastr THEN
          BEGIN
          &fl _ flntadr(auxwrk.stfile);
          fl.flnoclos _ FALSE;
          END;
    % reset upper/lower case translation %
        CASE nldevice OF
          =dev33, =dev35, =offline:  initch(nldevice);
          ENDCASE;
    RETURN;
    END.
%.....Window locating routines.....%
  (lcfile)  % CL: ;  find the file displayed in the current window %
  PROCEDURE  % => fileno <INTEGER> %;                                       16A
    % Procedure description
        FUNCTION
          find the file displayed in the current window
        ARGUMENTS
          none
        RESULTS
          fileno: the file number of the csp of the display area of
          the current window
        NON-STANDARD CONTROL
          none
        GLOBALS
          none
        %
    % Declarations %
        LOCAL stid;
    % Get file number%
        RETURN(dspfile(cwndow));
    % Return %
```

```
      %return above%
   END.

(lccsp)   % CL: ;  get Current Statement Pointer for current window
%
PROCEDURE % => csp <TEXT POINTER>, ccnt <INTEGER> %;                   16B
   % Procedure description
      FUNCTION
         returns the current statement pointer of the display area
         of the current window
      ARGUMENTS
         none
      RESULTS
         csp:  current statement pointer
      NON-STANDARD CONTROL
         SIGNALS generated
            err($"Display area does not exist - lccsp") if display
            area does not exist
      GLOBALS
         cwndow
      %
   % Declarations %
      LOCAL da REF;
   %Get da for current window%
      IF NOT(&da _ dsparea(cwndow)) THEN
         err($"Display area does not exist - lccsp");
   % Return %
      RETURN(da.dacsp, da.daccnt);
   END.

%
(lcda) %%find the window containing the cursor during the last input
control character%%
PROCEDURE;                                                             16D
   LOCAL dacords;
   IF nlmode = typewriter THEN RETURN((&tda-$dpyarea)/dal + 1, 0);
   dacords _ lccords();
   RETURN(findda(dacords : dacords.xcord, dacords.ycord), dacords);
   END.

%
(lda)  % returns address of Display Area descriptor for the display
area where the cursor was the Last time a character was input %
PROCEDURE;                                                             16F
   %GLOBAL cwndow%
   RETURN (dsparea(cwndow)); END.

(dspfile) %Get file in display area record for window id%
PROCEDURE (wid);                                                       16C
   LOCAL da REF;
   IF NOT (&da _ dsparea(wid)) THEN
      err($"Display area does not exist - dspfile");
   RETURN(da.dacsp.stfile);
   END.

(dsparea) %Get display area record for window id -- returns zero if
```

```
display area is not allocated%
PROCEDURE (wid);                                                       16H
   LOCAL wa REF;
   &wa _ findwa(wid);
   RETURN(wa.widdarea);
   END.

(findwa)  %Get window area address %
PROCEDURE (wid % => wa <address> %);                                   16I
   % Procedure description
      FUNCTION
         Get the window area address of the window whose window
         identifier is wid.  If this is DNLS and there is only one
         display window, use the display window even if its window
         identifier is not wid.
      ARGUMENTS
         wid:  front-end window id
      RESULTS
         wa:  address of window area record for wid or zero if
         matching wid not found
      NON-STANDARD CONTROL
         SIGNALS generated
            err($"Place cursor mark in display window") if window
            id not found in any window area
      GLOBALS
         wndwarea, wal, wacnt,
      %
   % Declarations %
      LOCAL end, wa REF,dwa REF, dwndwct _ 0;
   %Search through window areas for matching window identifier%
      &wa _ $wndwarea;
      end _ $wndwarea + wal*(wacnt+1);
      WHILE &wa < end AND (wa.widindex # wid OR (NOT wa.widexis)) DO
         BEGIN
         IF wa.widdarea AND wa.wtype = rwtype THEN
            BEGIN  %display window that is currently in use%
            BUMP dwndwct;
            &dwa _ &wa;  %save address in case only 1 display%
            END;
         &wa _ &wa + wal;
         END;
      IF &wa >= end THEN
         BEGIN  %matching window identifier not found%
         IF dwndwct = 1 THEN  %only 1 display window%
            &wa _ &dwa  %use the display window%
         ELSE err($"Place cursor mark in display window");
         END;
   % Return %
      RETURN(&wa);
   END.

(newda) %allocate new display area%
PROCEDURE;                                                             16J
   LOCAL entry, end;
   REF entry;
   IF dacnt NOT IN [0, damax] THEN
```

```
         err($"Illegal dacnt, newda");
      end _ (&entry _ $dpyarea-dal) + dacnt*dal;
      UNTIL (&entry _ &entry + dal) > end DO
         IF NOT entry.daexis THEN EXIT;
      IF &entry > end THEN
         BEGIN
         IF dacnt = damax THEN
            err($"Too many display areas");
         dacnt _ dacnt + 1;
         END;
      entry.daexis _ TRUE;
      entry.dawid _ 0;  %not associated with window yet%
      entry.dalink _ newfrring(frrsize);  %get return rings%
      entry.dacsp _ endfil;
      entry.daccnt _ 1;
      entry.daempty _ TRUE;
      entry.daauxiliary _ FALSE; %assume to be used for display
      purposes%
      entry.dastrt _ 0; %no string table allocated yet%
      RETURN(&entry);
      END.

   (newwa) PROCEDURE;  %allocate new window area%                      16K
      LOCAL entry REF, end;
      IF wacnt NOT IN [0, wamax] THEN
         err($"Illegal wacnt, wal");
      end _ (&entry _ $wndwarea-wal) + wacnt*wal;
      UNTIL (&entry _ &entry + wal) > end DO
         IF NOT entry.widexis THEN EXIT;
      IF &entry > end THEN
         BEGIN
         IF wacnt = wamax THEN
            err($"Too many display areas");
         wacnt _ wacnt + 1;
         END;
      entry.widexis _ TRUE;
      RETURN (&entry);
      END.
   (copyda)   % CL:GB ;  copy contents of da %
   PROCEDURE (dafrom REF, dato REF);                                   16L
      % Procedure description
         FUNCTION
            Copy contents of da excluding return ring, string table and
            frozen statement chain.
         ARGUMENTS
            dafrom:  display area address from which to copy data
            dato:  display area address into which to copy data
         RESULTS
            proc-value
         NON-STANDARD CONTROL
            none
         GLOBALS
            none
         %
      % Declarations %
         LOCAL end, dadest REF, frr REF;
      % Copy data %
         &dadest _ &dato;  %save original value%
         &frr _ dato.dalink;
         %copy da entry%
            end _ &dafrom + dal;
            DO
               BEGIN
               dato _ dafrom;
               BUMP &dato;
               END
            UNTIL (&dafrom _ &dafrom + 1) = end;
      % Restore return ring and zap string table and frozen statement
      chain %
         dadest.dalink _ &frr;
         dadest.dastrt _ dadest.dacurallo _ dadest.dafrzl _ 0;
      % Return %
         RETURN;
      END.

   (copywa)   % CL:GB ;  copy contents of wa %
   PROCEDURE (wafrom REF, wato REF);                                16M
      % Procedure description
         FUNCTION
            Copy contents of wa.
         ARGUMENTS
            wafrom:  window area address from which to copy wato
            wato:  window area address into which to copy wato
         RESULTS
            proc-value
         NON-STANDARD CONTROL
            none
         GLOBALS
            none
         %
      % Declarations %
         LOCAL end;
      %copy wa entry%
         end _ &wafrom + wal;
         DO
            BEGIN
            wato _ wafrom;
            BUMP &wato;
            END
         UNTIL (&wafrom _ &wafrom + 1) = end;
      % Return %
         RETURN;
      END.

%.....process Front-end Lists Routines.....%
   (setwa)   % CL:LB ;  Set up wa from get-windows LIST %
   PROCEDURE (wa REF, wlist REF LIST % => %);                       17A
      % Procedure description
         FUNCTION
            Process a window LIST set from the front-end in a
            get-windows call.  "setwa" is called for each window
            defined by the front-end.  Set up the corresponding
         back-end window data structure in the block "wa".
      ARGUMENTS
         wa:  address of a block in window area.  set by "setwa".
         wlist:  window-list from front-end.  LIST(owning-window-id,
         window-id, type,diag-coords, window-att)
      RESULTS
         proc-value
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
         %
   % Declarations %
      LOCAL lstptr REF;
   % Set up window-id, owning window-id, type and zero out da
   address%
      wa.widdarea _ 0;  %No display area yet%
      wa.widindex _ ELEM #wlist#[2];
      wa.widowner _ ELEM #wlist#[1];
      wa.wtype _ ELEM #wlist#[3];
   %Process diagonal coordinate list%
      &lstptr _ ELEM #wlist#[4];
      wa.wuplx _ ELEM #lstptr#[1];
      wa.wuply _ ELEM #lstptr#[2];
      wa.wlrx _ ELEM #lstptr#[3];
      wa.wlry _ ELEM #lstptr#[4];
   %Process window-att list for priority and default string
   attributes%
      &lstptr _ ELEM #wlist#[5];
      wa.wipriority _ ELEM #lstptr#[1];
      %Process string-att list.  Ignore coordinates (element 1)%
         &lstptr _ ELEM #lstptr#[2];
         wa.wiatt _ ELEM #lstptr#[2];
         wa.wiselector _ ELEM #lstptr#[3];
   % Return %
      RETURN;
   END.

%.....return ring routines.....%
  (ckrrings) PROC; %check consistency of return rings%                18A
     LOCAL da, end, frr, srr, frrend;
     REF da, frr, srr;
     end _ (&da _ $dpyarea) + dal*dacnt;
     DO IF da.daexis THEN
        BEGIN
        IF NOT (&frr _ da.dalink) THEN
           werr($"return ring error: dalink empty");
        IF NOT frr.frhexis THEN
           werr($"return ring error: FRR empty");
        frrend _ &frr + frr.frhlast*frrelen + frrhlen;
        FOR &frr _ &frr + frrhlen UP frrelen UNTIL >=frrend DO
           IF frr.frexis THEN
              BEGIN
              IF NOT (&srr _ frr.frsrring) THEN
                 werr($"return ring error: FRSRING field empty");
              IF NOT srr.srhexis THEN
              werr($"return ring error: SRR empty");
           END;
      END
   UNTIL (&da _ &da + dal) >= end;
   RETURN;
   END.

(newfrring)  % allocate a new file return ring %
 PROCEDURE (size);                                                     18B
   LOCAL frr;
   REF frr;
   IF size NOT IN [1,frrmax] THEN err($"Illegal size requested for
   file return ring");
   %get frr block%
      &frr _ getblk((size*frrelen)+frrhlen, $dspblk);
      IF NOT &frr THEN err($"Insufficient space for new file return
      ring");
   %init frr block%
      frr.frhlast _ size-1;
      frr.frhexis _ TRUE;
   RETURN(&frr);
   END.
(freefrring)  %free file return ring%
PROCEDURE (frh);                                                       18C
   LOCAL i, end, frr;
   REF frr, frh;
   IF NOT frh.frhexis THEN err($"Illegal file return ring");
   end _ &frh + frrhlen + (frrelen*frh.frhlast);
   %free statement return rings%
      FOR &frr _ &frh +frrhlen UP frrelen UNTIL > end DO
         IF frr.frexis THEN freesrring(frr.frsrring);
   end _ end + frrelen-1;
   %zero block as a precaution%
      FOR i _ &frh UP UNTIL > end DO [i] _ 0;
   %free block%
      freeblk(&frh, $dspblk);
   RETURN;
   END.

(pushfrring)  %push file return ring%
PROCEDURE (frr, filename, fileno);                                     18D
   LOCAL srr, fre;
   REF frr, fre, srr, filename;
   IF NOT frr.frhexis THEN err($"Illegal file return ring");
   % zero file number of current top SRRING %
      &fre _ &frr + frrhlen + (frrelen*frr.frhtop);
      IF fre.frexis THEN
         BEGIN
         &srr _ frr.frsrring;
         srr.srhfileno _ 0;
         END;
   %increment top-of-ring pointer%
      frr.frhtop _ IF frr.frhtop >= frr.frhlast THEN 0 ELSE
      frr.frhtop+1;
   %get address of new top entry%
      &fre _ &frr + frrhlen + (frrelen*frr.frhtop);
```

```
    %if there is an old statement return ring, free it%
       IF fre.frexis AND fre.frsrring THEN freesrring(fre.frsrring);
    %allocate a statement return ring%
       fre.frsrring _ &srr _ newsrring(srrsize);
       fre.frexis _ TRUE;
    %allocate a string for file name%
       srr.srhfname _ getstring(filename.L, $dspblk);
    %store away file name and number%
       *[srr.srhfname]* _ *filename*;
       srr.srhfileno _ fileno;
    RETURN;
    END.
(readfrring) %Read file return ring entry.%
PROCEDURE (frr, index);                                              18E
    % case index of
       = 0: read top entry
       > 0 decrement over index good entries and return contents
       endcase;%
    %returns address of file name string (READ ONLY) and address of
    statement return ring%
    % ------------------- %
    LOCAL i, stid, j, last, base, count, fre, srr;
    REF frr, fre, srr;
    IF NOT frr.frhexis THEN err($"Illegal file return ring");
    %initialization%
       j _ frr.frhtop;
       last _ frr.frhlast;
       base _ &frr + frrhlen;
       IF index < 0 THEN
          err($"Illegal index in readfrring");
       count _ 0;  %used to detect empty ring%
       &fre _ base + (frrelen*j);  %current top%
    FOR i _ index MOD (last+1) DOWN UNTIL <= 0 DO
       BEGIN
       %move one entry%
          j _ IF j <= 0 THEN last ELSE j -1;
          &fre _ base + (frrelen*j);
       %continue moving until find valid entry%
          UNTIL fre.frexis DO
             BEGIN
             j _ IF j <= 0 THEN last ELSE j -1;
             &fre _ base + (frrelen*j);
             IF (count _ count+1) > last THEN
                err($"no entries in statement return ring --
                backfrring");
             END;
          count _ 0;
       END;
    IF NOT fre.frexis THEN ABORT(frremptyentry, $"current entry empty
    in readfrring");
    &srr _ fre.frsrring;
    IF NOT srr.srhexis THEN
       err($"Illegal statement return ring detected in readfrring");
    RETURN(srr.srhfname, &srr);
    END.
```

```
(frrlength) %return number of entries in the file return ring.%
PROCEDURE (frr);                                                        18F
   LOCAL i, j, k, last, base, count, fre, srr, top;
   REF frr, fre, srr;
   % ------------------- %
   IF NOT frr.frhexis THEN
      err($"Illegal file return ring in frrlength");
   %initialization%
      j _ top _ frr.frhtop;
      last _ frr.frhlast;
      base _ &frr + frrhlen;
      &fre _ base + (frrelen*top);  %current top%
      IF fre.frexis THEN k _ 0 ELSE k _ 1;  %see end of proc%
      count _ 0;
      UNTIL fre.frexis DO
         BEGIN
         j _ IF j <= 0 THEN last ELSE j -1;
         &fre _ base + (frrelen*j);
         IF (count _ count+1) > last OR j = top THEN
            RETURN(-1);
         END;
   FOR i _ 0 UP UNTIL > last DO
      BEGIN
      %move one entry%
         j _ IF j <= 0 THEN last ELSE j -1;
         &fre _ base + (frrelen*j);
         IF j = top THEN EXIT LOOP;
      %continue moving until find valid entry%
         count _ 0;
         UNTIL fre.frexis DO
            BEGIN
            j _ IF j <= 0 THEN last ELSE j -1;
            &fre _ base + (frrelen*j);
            IF j = top THEN EXIT LOOP 2;
            IF (count _ count+1) > last THEN
               RETURN(-1);
            END;
      END;
   i _ i + k;  %to take account of the different way readfrring acts
   if the top entry is empty; need the count of entries to include
   that one%
   RETURN(i);
   END.

(upisfnm) %change file name ofn to nfn in all file return rings%
PROCEDURE (ofn, nfn);                                                   18G
   %ofn,   address of string containing complete old file name%
   %nfn,   address of string containing complete new file name%
   LOCAL da, end, i, str, len, srr;
   REF ofn, nfn, da, str, srr;
   %-----------------------%
   end _ $dpyarea + dacnt*dal;
   INVOKE (sigloop, rptloop);
   DISABLE (sigloop);
   FOR &da _ $dpyarea UP dal UNTIL > end DO
      IF da.daexis THEN
```

```
         BEGIN
         len _ frrlength(da.dalink);
         i _ -1;
         (rptloop):                                                 18G9A4
         UNTIL (i _ i+1) > len DO
            BEGIN
            ENABLE (sigloop);
            &str _ readfrring(da.dalink, i: &srr);
            DISABLE (sigloop);
            IF &str AND (*str* = *ofn*) THEN
               BEGIN
               IF str.M < nfn.L THEN
                  BEGIN
                  freestring(&str, $dspblk);
                  &str _ getstring(nfn.L, $dspblk);
                  srr.srhfname _ &str;
                  END;
               *str* _ *nfn*;
               END;
            END;
         END;
   DROP( ALL);
   RETURN;


   (sigloop) CATCHPHRASE();
      BEGIN
      CASE SIGNALTYPE OF
         = notetype :  NULL;
         = helptype :  NULL;
         = aborttype :
            BEGIN
            DISABLE(sigloop);
            IF SIGNAL = frremptyentry THEN TERMINATE;
            END;
         ENDCASE;
      CONTINUE;
      END;
   END.

(mkdelfrr) PROCEDURE (frr, filename);                               18H
   %Mark file return ring entries for deleted files.  Set frexis
   field to FALSE, and free associated statement return ring%
   LOCAL fre, srr, frrfname;
   REF frr, fre, srr, frrfname, filename;
   FOR &fre _ &frr + frrhlen UP UNTIL > &frr + frr.frhlast + frrhlen
   DO
      IF fre.frexis THEN
         BEGIN
         &srr _ fre.frsrring;
         &frrfname _ srr.srhfname;
         IF *frrfname* = *filename* THEN
            BEGIN
            fre.frexis _ FALSE;
            freestring(&srr, dspblk);
            END;
         END;
      RETURN;
      END.
(newsrring) %allocate a new statement return ring%
PROCEDURE (size);                                                    18I
      LOCAL srr;
      REF srr;
      IF size NOT IN [1,srrmax] THEN err($"Illegal size requested for
      statement return ring");
      %get srr block%
         &srr _ getblk((size*srrelen)+srrhlen, $dspblk);
         IF NOT &srr THEN
            err($"Insufficient space for new statement return ring");
      %init srr block%
         srr.srhlast _ size-1;
         srr.srhexis _ TRUE;
      RETURN(&srr);
      END.
(freesrring) %free statement return ring%
PROCEDURE (srh);                                                     18J
      LOCAL i, end;
      REF srh;
      IF NOT srh.srhexis THEN err($"Illegal statement return ring");
      %free file name string%
         IF srh.srhfname THEN freestring(srh.srhfname, $dspblk);
         srh.srhfname _ 0;
      %zero block as a precaution%
         end _ &srh + srrhlen + (srh.srhlast*srrelen) + srrelen-1;
         FOR i _ &srh UP UNTIL > end DO [i] _ 0;
      %free block%
         freeblk(&srh, $dspblk);
      RETURN;
      END.
(copysrring) %copy statement return ring%
PROCEDURE (fromsrr, tosrr);                                          18K
      %does not change file name string or file number in tosrr.
      Copies body of fromsrr to tosrr and set top-of-ring in tosrr to
      correspond with fromsrr.%
      LOCAL i, end;
      REF fromsrr, tosrr;
      IF NOT fromsrr.srhexis OR NOT tosrr.srhexis THEN
         err($"Illegal statement return ring in copysrring");
      %copy block%
         end _ srrhlen + (tosrr.srhlast*srrelen) + srrelen-1;
         FOR i _ srrhlen UP UNTIL > end DO tosrr[i] _ fromsrr[i];
         tosrr.srhtop _ fromsrr.srhtop;
      RETURN;
      END.
(pushsrring) %push contents onto statement return ring%
PROCEDURE (srr, stid, cc, vs1, vs2);                                 18L
      %increment and store%
      LOCAL sre; REF sre;
      REF srr;
      %verify file number%
         IF srr.srhfileno AND stid.stfile NOT= srr.srhfileno THEN
            err($"file numbers do not match in pushsrring");
```

```
   %increment top-of-ring pointer%
      srr.srhtop _ IF srr.srhtop >= srr.srhlast THEN 0 ELSE
      srr.srhtop+1;
   %get address of top entry%
      &sre _ &srr + srrhlen + (srrelen*srr.srhtop);
   %store new values%
      sre.srpsid _ stid.stpsid;
      sre.srcc _ cc;
      sre.srvs1 _ vs1;
      sre.srvs2 _ vs2;
      sre.srexis _ TRUE;
   RETURN;
   END.
(storesrring) %store contents onto statement return ring.  INDEX
has the same semantics as in readsrring.%
PROCEDURE (srr, index, stid, cc, vs1, vs2);                          18M
   LOCAL sre, i, j, last, base, count;
   REF srr, sre;
   %verify file number%
      IF srr.srhfileno AND stid.stfile NOT= srr.srhfileno THEN
         BEGIN
         dismes(1, $"Bad file number in stmt return ring; advise you
         ^C, reset, and call NLS.");
         RETURN;
         END;
   CASE index OF   % pick out the srr entry to update %
      %> srr: &sre _ index;  caller specified entry%
      = 0:  &sre _ &srr + srrhlen + srrelen*srr.srhtop;  %top%
      ENDCASE   %use index to entry as in readsrring%
         BEGIN
         j _ srr.srhtop;
         last _ srr.srhlast;
         base _ &srr + srrhlen;
         count _ 0;  %used to detect empty ring%
         IF index < 0 THEN err($"Illegal index in storesrring");
         &sre _ base + (srrelen*j);
         FOR i _ index MOD (last+1) DOWN UNTIL <= 0 DO
            BEGIN
            %move one entry%
               j _ IF j <= 0 THEN last ELSE j -1;
               &sre _ base + (srrelen*j);
            %continue backing up until find valid entry%
               UNTIL sre.srexis DO
                  BEGIN
                  j _ IF j <= 0 THEN last ELSE j -1;
                  &sre _ base + (srrelen*j);
                  IF (count _ count+1) > last THEN
                     err($"no entries in statement return ring --
                     storesrring");
                  END;
               count _ 0;
            END;
         IF NOT sre.srexis THEN
            err($"current entry empty in storesrring");
         END;
   %store new values%
```

```
      sre.srpsid _ stid.stpsid;
      sre.srcc _ cc;
      sre.srvs1 _ vs1;
      sre.srvs2 _ vs2;
      sre.srexis _ TRUE;
   RETURN;
   END.
(readsrring)  %Read statement return ring entry.%                    18N
   PROCEDURE (srr, index);
   % case index of
      = 0: read top entry
      > 0: decrement over index good entries and return contents
      endcase;%
   %returns stid, cc, vs1, vs2%
   % ------------------ %
   LOCAL i, stid, j, last, base, count, inc, sre;
   REF srr, sre;
   IF NOT srr.srhexis THEN err($"Illegal statement return ring");
   %initialization%
      stid _ 0;
      stid.stfile _ srr.srhfileno;
      j _ srr.srhtop;
      last _ srr.srhlast;
      base _ &srr + srrhlen;
      count _ 0;  %used to detect empty ring%
      IF index < 0 THEN err($"Illegal index in readsrring");
      &sre _ base + (srrelen*j);
   %move through the ring%
      FOR i _ index MOD (last+1) DOWN UNTIL <= 0 DO
         BEGIN
         %move one entry%
            j _ IF j <= 0 THEN last ELSE j -1;
            &sre _ base + (srrelen*j);
         %continue backing up until find valid entry%
            UNTIL sre.srexis DO
               BEGIN
               j _ IF j <= 0 THEN last ELSE j -1;
               &sre _ base + (srrelen*j);
               IF (count _ count+1) > last THEN
                  err($"no entries in statement return ring --
                  readsrring");
               END;
            count _ 0;
         END;
   IF NOT sre.srexis THEN
      err($"current entry empty in readsrring");
   stid.stpsid _ sre.srpsid;
   RETURN(stid, sre.srcc, sre.srvs1, sre.srvs2);
   END.

(srrlength)  %return number of entries in statement return ring.%
PROCEDURE (srr);                                                      18O
   LOCAL i, j, last, base, count, sre, top;
   REF srr, sre;
   % ------------------ %
   IF NOT srr.srhexis THEN
```

```
         err($"Illegal statement return ring in srrlength");
      %initialization%
         j _ top _ srr.srhtop;
         last _ srr.srhlast;
         base _ &srr + srrhlen;
         &sre _ base + (srrelen*top);
         count _ 0;
         UNTIL sre.srexis DO
            BEGIN
            j _ IF j <= 0 THEN last ELSE j -1;
            &sre _ base + (srrelen*j);
            IF (count _ count+1) > last OR j = top THEN
               RETURN(0);
            END;
      %count valid entries in the ring%
         FOR i _ 0 UP UNTIL > last DO
            BEGIN
            %move one entry%
               j _ IF j <= 0 THEN last ELSE j -1;
               &sre _ base + (srrelen*j);
               IF j = top THEN EXIT LOOP;
            %continue backing up until find valid entry%
               count _ 0;
               UNTIL sre.srexis DO
                  BEGIN
                  j _ IF j <= 0 THEN last ELSE j -1;
                  &sre _ base + (srrelen*j);
                  IF j = top THEN EXIT LOOP 2;
                  IF (count _ count+1) > last THEN
                     RETURN(0);
                  END;
            END;
      RETURN(i);
      END.

FINISH of frontend



%... old stuff.  Moved to here by ROM ...%
   (ctlquit) PROCEDURE;  %shut off control stuff%                    20A
      %close control file (if one is being used), reinitialize control
      jfn's, and reset subsystem name%
      %--------------%
      LOCAL jfn;
      &rawchr _ IF nldevice # devlproc THEN $getchar ELSE
         IF altinp.L THEN $lpaltgetchar ELSE $lpgetchar;
         %set lowest level input routine to be simple get-a-character
         routine%
      IF inpwatchjfn THEN jfn _ inpwatchjfn  %recording session%
      ELSE
         IF inpjfn THEN jfn _ inpjfn %playback session%
         ELSE RETURN;
      IF NOT sysclose(jfn, $lit) THEN dismes(2, $lit);
      inpwatchjfn _ inpjfn _ 0;
      R1 _ nlssbn _ getsbn($"DNLS");
      %if control environment ever run from tty or net, should be
      smart enough to figure out real name%
   !JSYS setnm;
   RETURN;
   END.
```

```
(ckident)       <nine, identsupport, 02660>       PROCEDURE         1C1
(ckilmem)       <nine, identsupport, 03370>       PROCEDURE         1E1
(ckipmem)       <nine, identsupport, 02476>       PROCEDURE         1E2
(geticapability) <nine, identsupport, 0675>       PROCEDURE         1D12
(getidelivery) <nine, identsupport, 0701>         PROCEDURE         1D13
(getiend)       <nine, identsupport, 03642>       PROCEDURE         1D6
(getifnt)       <nine, identsupport, 0492>        PROCEDURE         1D3
(getigrps)      <nine, identsupport, 03628>       PROCEDURE         1D5
(getihost)      <nine, identsupport, 0611>        PROCEDURE         1D9
(getiid)        <nine, identsupport, 0462>        PROCEDURE         1D1
(getinam)       <nine, identsupport, 0481>        PROCEDURE         1D2
(getinlhost)    <nine, identsupport, 0624>        PROCEDURE         1D14
(getinma)       <nine, identsupport, 0637>        PROCEDURE         1D10
(getiorg)       <nine, identsupport, 03073>       PROCEDURE         1D4
(getiuser)      <nine, identsupport, 0572>        PROCEDURE         1D8
(getpointer)    <nine, identsupport, 03604>       PROCEDURE         1E8
(identinterp)   <nine, identsupport, 03489>       PROCEDURE         1B1
(identquery)    <nine, identsupport, 03731>       PROCEDURE         1B2
(idflsearch)    <nine, identsupport, 03255>       PROCEDURE         1E3
(idpgrporg)     <nine, identsupport, 01776>       PROCEDURE         1E6
(idpind)        <nine, identsupport, 01764>       PROCEDURE         1E5
(idpiname)      <nine, identsupport, 01746>       PROCEDURE         1E4
(idelivery)     <nine, identsupport, 0246>        PROCEDURE         1D11
(namesearch)    <nine, identsupport, 03083>       PROCEDURE         1E7
(ognxt)         <nine, identsupport, 03566>       PROCEDURE         1D7
(orgrptst)      <nine, identsupport, 03609>       PROCEDURE         1E9
(stnamcap)      <nine, identsupport, 01552>       PROCEDURE         1C2
```

```
< NINE, IDENTSUPPORT.NLS;9, >, 13-Apr-78 10:27 LLG ;;;;
FILE identsupport % (arcsubsys, 1109,) (relnine,identsupport.rel,) %
   % declarations....%                                              1A
      DECLARE EXTERNAL lname=1, strinname=2, idchr=3, afgptyp=4;
      DECLARE EXTERNAL STRING nwidstr = "NEWIDS", nullfield = "NONE";
   % argument conversion support %                                  1B
      (identinterp)   % CL: ;  interpret identlist string %
      PROCEDURE (idstr REF % => &cnvidstr %);                       1B1
         % Procedure description
            FUNCTION
               Convert a typed-in ident list, which may contain things
               like .username or .usern..., into a list of idents only.
                No expansion of group idents is done.  This may result
               in interaction with the user to find a single ident for
               entries entered as username.
            ARGUMENTS
               idstr--REF-addr of string containing IDENTLIST
            RESULTS
               &cnvidstr--addr of new string containing converted list.
            NON-STANDARD CONTROL
               none
            GLOBALS
               none
            %
         % Declarations %
            LOCAL cnvidstr REF;
            LOCAL TEXT POINTER z1, z2;
            LOCAL STRING work[50];
         % check for no string, allocate return string %
            IF idstr.L = empty THEN RETURN(FALSE);
            &cnvidstr _ getstring(2000, $dspblk);
            *cnvidstr* _ *idstr*;
         % parse the identlist %
            % initialize %
               FIND SF(*cnvidstr*) ^z2;
               WHILE (FIND z2 ['"/'.] < CH > ^z1) DO
                  BEGIN
                  IF FIND '" THEN FIND ['"/ENDCHR] ^z2
                  ELSE FIND CH $(LD / '' / SP / '-) $'. ^z2;
                  *work* _ z1 z2;
                  IF work.L THEN
                     BEGIN
                     IF identquery($work) THEN %user query was
                     sucessful%
                        ST z1 z2 _ *work*
                     ELSE
                        BEGIN
                        FIND z2 > $(SP/',) ^z2;
                        ST z1 z2 _ NULL;
                        END;
                     FIND z1 ^z2;
                     END;
                  END;
         % Return %
            RETURN(&cnvidstr);
         END.
```

```
(identquery)   % CL: ;  Convert Identlist member to valid ident %
PROCEDURE (idstr REF);                                           1B2
   % Procedure description
      FUNCTION
         Convert an ident list element, which may be something
         like .username or .usern..., into a valid idents.  No
         expansion of group idents is done.  This may result in
         interaction with the user to find a single ident for an
         element entered as username.
      ARGUMENTS
         idstr--REF-addr of string containing IDENTLIST
      RESULTS
         none
      NON-STANDARD CONTROL
         none
      GLOBALS
         none
      %
   % Declarations %
      LOCAL type;
      LOCAL TEXT POINTER tptr1, tptr2;
   % check for no string, allocate return string %
      IF idstr.L = empty THEN RETURN(FALSE);
   % check for special search characters %
      CASE *idstr*[1] OF
         ='.: %lastname search%
            BEGIN
            *idstr* _ *idstr*[2 TO idstr.L];
            CCPOS SE(*idstr*);  %check for incomplete name%
            IF READC = '. THEN
               BEGIN %partially specified last name, remove
               trailing periods%
               DO BUMP DOWN idstr.L UNTIL READC NOT= '.;
               type _ idchr;
               END
            ELSE type _ lname;
            stnamcap(&idstr);
            idflsearch(type, &idstr, idfno) ;
            END;
         ='": %search name fields of individuals, groups and
         orgs%
            BEGIN
            FIND SF(*idstr*) CH ^tptr1 ( ['"] ^tptr2 _tptr2 /
            SE(*idstr*) ^tptr2);
            *idstr* _ tptr1 tptr2;
            idflsearch(strinname, &idstr, idfno) ;
            END;
         = SP: %deblank front of string%
            BEGIN
            *idstr* _ *idstr*[2 TO idstr.L];
            REPEAT CASE;
            END;
         ENDCASE;
   % Return %
      RETURN;
      END.

% check idents against ident-file....%
   (ckident) %validate an ident and fetch entry if desired%
   PROCEDURE (identsr, retinfo, fileno);                              1C1
      LOCAL idstid, retval, stid, count;
      LOCAL TEXT POINTER tp1, tp2;
      REF identsr, retinfo, lgngrps;
      %This routine checks the validity of the ident in identsr, and
      returns the information concerning that person from  he
      systems ident file in the retinfo string.  If fileno is
      non-zero, it is assumed to be the file number of the ident
      file...otherwise, the ident file is opened%
      %If a valid ident, also returns the stid of the statement
      which corresponds to the ident.%

      IF identsr.L = empty THEN RETURN(FALSE);
      idstid _ orgstid;
      IF NOT fileno THEN
         BEGIN
         INVOKE (clsfile);
         idstid.stfile _ open(0, jflname($"identfile"));
         END
      ELSE
         idstid.stfile _ fileno;
      astruc(&identsr);
      IF (retval _ ((stid _ namelook(idstid, &identsr)) # endfil))
      THEN
         BEGIN
         IF NOT (FIND SF(stid) [")] $SP "LAST NAME") THEN
            BEGIN
            IF &retinfo > 0 THEN
               BEGIN
               *retinfo* _ SF(stid) SE(stid);
               IF NOT &lgngrps AND *identsr* = *initsr* THEN
                  BEGIN
                  getigrps (&retinfo, 0, $tp1, $tp2);
                  IF (count _ tp2 [1] - tp1 [1]) THEN
                     BEGIN
                     &lgngrps _ getstring (count, $dspblk);
                     *lgngrps* _ tp1 tp2;
                     END
                  ELSE &lgngrps _ $" ";
                  END;
               END;
            END
         ELSE retval _ FALSE;
         END;
      IF NOT fileno THEN close(idstid.stfile := 0);
      RETURN (retval, stid);
      (clsfile) CATCHPHRASE ;                                         1C10
         BEGIN
         CASE SIGNALTYPE OF = aborttype:
            IF idstid.stfile THEN close(idstid.stfile := 0);
            ENDCASE;
         CONTINUE;
         END;
      END.

   (stnamcap) %Capitalize first letter of each word in a string%
   PROCEDURE (string);                                              1C2
      LOCAL char, count;
      REF string;
      %-----------------%
      count _ 1;
      CASE (char _ *string*[count]) OF
         IN ['a, 'z]:  *string*[count] _ char - 40B;
         =SP:
            BEGIN
            *string* _ *string*[count+1 TO string.L];
            REPEAT CASE;
            END;
         ENDCASE;
      WHILE (count _ count + 1) < string.L
         DO
         IF *string*[count] = SP
         AND (char _ *string*[count + 1]) IN ['a, 'z] THEN
            BEGIN
            *string*[count + 1] _ char - 40B;
            count _ count + 1;
            END;
      RETURN;
      END.

% retrieve fields from entry in ident-file....%
   (getiid) %get IDENT field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        1D1
      LOCAL TEXT POINTER lptr1, lptr2;
      REF entrystr;
      FIND SF(*entrystr*) $(SP/TAB) '( $NP ^lptr1 [')] <CH $NP>
      ^lptr2;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN (TRUE);
      END.

   (getinam) %get NAME field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        1D2
      LOCAL TEXT POINTER lptr1, lptr2;
      REF entrystr;
      IF orgrptst(&entrystr, 0) THEN
         FIND SF(*entrystr*) 2[EOL] $SP ^lptr1
      ELSE
         FIND SF(*entrystr*) [EOL] $SP ^lptr1;
      FIND lptr1 [EOL] < $NP ^lptr2 >;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN (TRUE);
      END.

   (getifnf) %get NAME field (FIRST NAME FIRST) field from an
   individual ident entry%
   PROCEDURE (entrystr, fldstr); %first name first%                 1D3
      LOCAL grpflg;
      LOCAL TEXT POINTER lptr1, lptr2, lptr3, lptr4;
      REF entrystr, fldstr;
      IF (grpflg _ orgrptst(&entrystr, 0)) THEN
         FIND SF(*entrystr*) 2[EOL]
      ELSE
         FIND SF(*entrystr*) [EOL];
      FIND $SP ^lptr1 ^lptr2 ^lptr3 [EOL] < $NP ^lptr4 >;
      IF NOT grpflg THEN FIND BETWEEN lptr1 lptr4 ([',] $NP ^lptr3 <
      $NP CH $NP ^lptr2);
      *fldstr* _ lptr3 lptr4, SP, lptr1 lptr2;
      RETURN (TRUE);
      END.

   (getiorg) %get ORGANIZATION field from an individual ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        1D4
      LOCAL TEXT POINTER lptr1, lptr2;
      REF entrystr;
      IF orgrptst(&entrystr, 0)
      OR NOT
         (FIND SF(*entrystr*) [')] $SP ^lptr1 [';/EOL] < CH $NP>
         ^lptr2 ) THEN
            FIND SE(*entrystr*) ^lptr1 ^lptr2;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN (TRUE);
      END.

   (getigrps) %get GROUPS field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        1D5
      LOCAL strptr;
      LOCAL TEXT POINTER lptr1, lptr2;
      LOCAL retval;
      REF entrystr;
      strptr _ getpointer(&entrystr);
      retval _ FALSE;
      IF NOT FIND SF(strptr) ["Groups:"] $NP ^lptr1 [';] < CH $NP >
      ^lptr2 THEN
         getiend (&entrystr, $lptr1, $lptr2)
      ELSE retval _ TRUE;
      stptset (fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN (retval);
      END.

   (getiend) %find the end of an ident entry%
   PROCEDURE (entrystr, ptr1, ptr2);                                1D6
      %fixed so entrystr may also be a-string or stid%
      LOCAL strptr;
      REF entrystr, ptr1, ptr2;
      strptr _ getpointer(&entrystr);
      IF NOT
         (FIND SF(strptr)
            ["Comments:"] EOL < 10 CH > ^ptr1 ^ptr2) THEN
               FIND SE(strptr) ^ptr1 ^ptr2;
      RETURN;
      END.

   (ognxt) %search list of idents for next group or org.%
```

```
PROCEDURE(l1, l2, idstid);                                         1D7
   %start at l1, return TRUE with text pointers l1 and l2 around
   ident on success.  return FALSE failure.  idstid must point
   into open identfile%
   %-----------------%
   LOCAL STRING tidstr[40];
   REF l1, l2;
   LOOP
      BEGIN
      IF NOT (FIND l2 $(SP / TAB / ',) ^l1 1$( LD / '- ) ^l2 )
      THEN RETURN(0);
      *tidstr* _ l1 l2;
      IF (idstid _ namelook(idstid, $tidstr)) # endfil AND
      orgrptst(idstid, 0) THEN RETURN(TRUE);
      END;
   END.

(getiuser) %get USER (NLS delivery) field from an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                          1D8
   LOCAL TEXT POINTER lptr1, lptr2;
   LOCAL retval;
   REF entrystr;
   retval _ FALSE;
   IF NOT
      (FIND SF(*entrystr*) ["User:"]
         $NP  ^lptr1 [';] < CH $NP > ^lptr2) THEN
            getiend(&entrystr, $lptr1, $lptr2)
   ELSE
      retval _ TRUE;
   stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
   RETURN(retval);
   END.

(getihost) %get NETWORK-DELIVERY HOST field from an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                          1D9
   LOCAL TEXT POINTER lptr1, lptr2;
   LOCAL retval;
   REF entrystr;
   retval _ FALSE;
   IF NOT
      (FIND SF(*entrystr*) ["Host:"] $NP ^lptr1 [';] < CH $NP >
         ^lptr2) THEN
            getiend(&entrystr, $lptr1, $lptr2)
   ELSE retval _ TRUE;
   stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
   RETURN(retval);
   END.

(getinma) %get NETWORK MAILBOX Address field from an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                          1D10
   LOCAL TEXT POINTER lptr1, lptr2;
   LOCAL retval;
   REF entrystr;
   retval _ FALSE;
   IF NOT
      (FIND SF(*entrystr*) ["Local Network Mailbox Address:"] $NP
         ^lptr1 [*; NP] < 2CH $NP >
            ^lptr2) THEN
               getiend(&entrystr, $lptr1, $lptr2)
      ELSE retval _ TRUE;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN(retval);
      END.

   (ldelivery) %get LOGICAL DELIVERY TYPE(S) for an ident entry%
   PROCEDURE (stid);                                              1D11
      %stid points to identification record.
         Returns record of delivery types indicated in/by record
         (see record definition 'deliverymode' in utilty for
         possible types)%
      LOCAL retval, oldelf, arcorgf, astrng;
      LOCAL TEXT POINTER z1, z2, z3, z4;
      LOCAL STRING tempsr[50];
      REF astrng;
      &astrng _ 0;
      INVOKE(catldel);
      &astrng _ getstring(2000, $dspblk);
      IF NOT stid.stastr THEN
         BEGIN
         *astrng* _ SF(stid) SE(stid);
         stid _ asrref(&astrng);
         END;
      getidelivery(stid.stadr, 0, $z1, $z2);
      retval _ 0;
      retval.delhc _ FIND BETWEEN z1 z2(["Hardcopy"]/["Hard Copy"]);
      retval.delnet _ FIND BETWEEN z1 z2(["Network"]/["Net Work"]);
      oldelf _ FIND BETWEEN z1 z2(["Online"]/["On-Line"]);
      IF retval = 0 OR oldelf THEN
         BEGIN
         getiorg(stid.stadr, $tempsr, 0, 0);
         arcorgf _ IF *tempsr* = "SRI-ARC" THEN TRUE ELSE FALSE;
         END;
      IF retval = 0 AND NOT oldelf THEN
         BEGIN
         %figure out a default delivery%
         %NLS users and ARC members get On-Line%
            IF arcorgf THEN oldelf _ TRUE
            ELSE
               BEGIN
               %everyone gets hard copy now%
                  retval.delhc _ TRUE;
               geticapability(stid.stadr, 0, $z1, $z2);
               oldelf _ FIND BETWEEN z1 z2(["NLS"]);
               END;
         END;
      IF oldelf THEN
         BEGIN
         oldelf _ IF arcorgf THEN 2 ELSE 1; %set default nlhost%
         getinlhost(stid.stadr, 0, $z3, $z4);
         IF z3[1] # z4[1] THEN
            BEGIN
            oldelf _ 0;
         IF FIND BETWEEN z3 z4 ([*arcistr*]) THEN oldelf _ oldelf
         .V 2;
         IF FIND BETWEEN z3 z4 ([*utilstr*]) THEN oldelf _ oldelf
         .V 1;
         IF FIND BETWEEN z3 z4 ([*nsastr*]) THEN oldelf _ oldelf
         .V 4;
         END;
      oldelf _ oldelf .A (CASE lhostn OF
         =archost: 2B;
         =utilhost: 1B;
         =nsahost: 4B;
         ENDCASE 0);
      IF oldelf THEN retval.delol _ TRUE;
      END;
   IF &astrng THEN freestring(&astrng:=0, $dspblk);
   RETURN(retval);
   (catldel) CATCHPHRASE();                                 1D11T
      BEGIN
      CASE SIGNALTYPE OF
         = aborttype :
            BEGIN
            DISABLE(catldel);
            IF &astrng THEN freestring(&astrng:=0, $dspblk);
            END;
         ENDCASE;
      CONTINUE;
      END;
   END.

(geticapability) %get CAPABILITIES field from an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                   1D12
   LOCAL retval;
   LOCAL TEXT POINTER lptr1, lptr2;
   REF entrystr;
   retval _ FALSE;
   IF NOT
      (FIND SF(*entrystr*) ["Capabilities:"] $NP ^lptr1 [";] < CH
      $NP >
         ^lptr2) THEN
            getiend(&entrystr, $lptr1, $lptr2)
   ELSE retval _ TRUE;
   stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
   RETURN(retval);
   END.

(getidelivery) %get DELIVERY-TYPES field from an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                   1D13
   LOCAL TEXT POINTER lptr1, lptr2;
   LOCAL retval;
   REF entrystr;
   retval _ FALSE;
   IF NOT
      (FIND SF(*entrystr*) ["Delivery:"] $NP ^lptr1 [";] < CH $NP
      >
         ^lptr2 ) THEN
            getiend(&entrystr, $lptr1, $lptr2)
      ELSE retval _ TRUE;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN(retval);
      END.

   (getinlhost) %get NLS-DELIVERY HOST field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        1D14
      LOCAL TEXT POINTER lptr1, lptr2;
      LOCAL retval;
      REF entrystr;
      retval _ FALSE;
      IF NOT
         (FIND SF(*entrystr*) ["NLS host:"] $NP ^lptr1 [';] < CH $NP
         >
            ^lptr2) THEN
               getiend(&entrystr, $lptr1, $lptr2)
      ELSE retval _ TRUE;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN(retval);
      END.

% miscellaneous utility routines %
   (ckilmem) %verify logical membership of an ident in an ident
   list%
   PROCEDURE (ident, l1, l2, openidfileno);                          1E1
      %----------%
      LOCAL outcome, idfileno, nident, istid;
      LOCAL STRING grpslist [1000], grpsident [35], idstr[35],
      checkedgrps [1000];
      LOCAL TEXT POINTER g1, g2, z1, z2, z3, z4;
      REF ident, l1, l2, lngrps, nident;
      %----------%
      idfileno _ openidfileno;
      %null list?%
         IF NOT (FIND l1 > (LD/'-) ^z1) OR z1 [1] > l2 [1] THEN
            RETURN (FALSE);
      %user appear explicitly in list?%
         *grpslist* _ l1 l2;
         astruc($grpslist);
         FIND SF(*grpslist*) ^z3 SE(*grpslist*) ^z4;
         IF ckipmem (&ident, $z3, $z4) THEN RETURN (TRUE);
      %get groups from access list%
         outcome _ FALSE;
         &nident _ 0;
         g1[1] _ g2[1] _ 1; %for initial test below%
         IF NOT idfileno THEN
            BEGIN
            idfileno _ open (0, jflname ($"Identfile"));
            INVOKE(ctch2close,ckilexit);
            END;
         istid _ orgstid; istid.stfile _ idfileno;
         *grpslist* _ NULL;
         FIND l1 ^z2;
         WHILE ognxt($z1, $z2, istid) AND z2[1] <= l2[1] DO
            *grpslist* _ *grpslist*, z1 z2, " ";
         IF grpslist.L = 0 THEN GOTO ckilexit; %no groups in access
      list%
      astruc($grpslist);
      FIND SF(*grpslist*) ^z3 SE(*grpslist*) ^z4 _z4;
   %locate ident's group list%
      IF *ident* = *initsr* AND &lgngrps THEN
         FIND SF(*lgngrps*) ^g1 SE(*lgngrps*) ^g2
      ELSE
         &nident _ &ident;
   *checkedgrps* _ NULL;
   WHILE g1[1] < g2[1] OR &nident DO
      BEGIN
      WHILE g1[1] < g2[1] DO
         BEGIN
         IF (FIND g1 > ^z1 1$( LD / '- ) ^z2 $(SP/TAB/',) ^g1)
         THEN
            BEGIN
            *idstr* _ z1 z2;
            IF ckipmem($idstr, $z3, $z4) THEN
               BEGIN
               outcome _ TRUE;
               GOTO ckilexit;
               END;
            *checkedgrps* _ *checkedgrps*, z1 z2, " ";
            END
         ELSE FIND g1 > CH ^g1; %skip over one char--avoids
         infinite loop in a bad list%
         END;
      IF &nident THEN %find super-groups of nident%
         BEGIN
         IF ckident (&nident:=0, 0, idfileno : istid) AND
         getigrps(istid, 0, $g1, $g2) AND g1[1] < g2[1] THEN
         REPEAT LOOP;
         END;
      IF checkedgrps.L # 0 THEN %break out last group to nident%
         BEGIN
         IF NOT (FIND SE(*checkedgrps*) < $NP ^z2 1$(LD/'-) ^z1)
         THEN err($"string err in ckilmem");
         *grpsident* _ z1 z2;
         &nident _ $grpsident;
         checkedgrps.L _ z1[1] - 1;
         END;
      END;
   DROP(ctch2close);
   (ckilexit): IF idfileno AND NOT openidfileno THEN                1E10
      close (idfileno := 0);
   RETURN (outcome);
   (ctch2close) CATCHPHRASE();                                       1E1Q
      BEGIN
      CASE SIGNALTYPE OF
         = aborttype :
            TERMINATE; % to ckilexit %
         ENDCASE;
      CONTINUE;
      END;
   END.
```

```
(ckipmem) %verify physical membership of an ident in an ident
list%
PROCEDURE (ident, l1, l2);                                          1E2
   %----------%
   LOCAL STRING listident [35];
   LOCAL TEXT POINTER lp1, lp2;
   REF ident, l1, l2;
   %----------%
   FIND l1 >;
   WHILE ((FIND $(SP/',) ^lp1 1$(LD/'-) ^lp2) AND (lp2 [1] <= l2
   [1])) DO
      BEGIN
      *listident* _ lp1 lp2;
      IF *ident* = *listident* THEN RETURN (TRUE);
      END;
   RETURN (FALSE);
   END.

(idflsearch) %search ident-file for last name or content-in-name%
PROCEDURE (type, idstr REF, fileno);                                1E3
   %Searches id file for individual's last name, first characters
   of individual's last names, or string, depending on type.
   Returns true or false depending on whether an ident is
   accepted.. If true, idstr contains the ident.%
   %------------%
   LOCAL  count, idstid, lnamstr, mode, outcome;
   LOCAL STRING newidstr[20], helpmess[100], work[500];
   LOCAL TEXT POINTER tptr1, tptr2;
   LOCAL LIST ans[2];
   idstid _ orgstid;
   IF fileno THEN  idstid.stfile _ fileno
   ELSE
      BEGIN
      idstid.stfile _ open(0, jflname($"identfile"));
      INVOKE(idsfail, frtn);
      END;
   CASE namesearch(&idstr, $newidstr, type, idstid.stfile) OF
      = 1 : %only one hit -- is it correct%
         BEGIN
         IF type = lname THEN
            BEGIN  % get help from user %
            *helpmess* _ "Is this the correct ", *idstr*, "?";
            typelit(nocawait, $helpmess);
            IF NOT (outcome _ helpfe(101, 0, 0, $ans)) THEN EXIT
            CASE;
            END
         ELSE
            BEGIN  % prompt is "Is this the correct one?" %
            IF NOT (outcome _ helpfe(102, 0, 0, $ans)) THEN EXIT
            CASE;
            END;
         IF #[#ans#[1]]#[1] THEN
            BEGIN  % user answered yes %
            *idstr* _ *newidstr*;
            *helpmess* _ "Ident ", *idstr*, " accepted";
            dismes(1, $helpmess);
```

```
              outcome _ TRUE;
              END
           ELSE outcome _ FALSE;
           END;
        > 1 : %more than one hit -- ask for correct one%
           BEGIN % prompt is "Type the correct IDENT:" %
           IF NOT (outcome _ helpfe(103, 0, 0, $ans)) THEN EXIT
           CASE;
           *idstr* _ *[[#[#ans#[1]]#[tppair]].RH]*;  % helpcall
           returns LSEL(#"TEXT") %
           astruc(&idstr);
           outcome _ idstr.L;
           END;
        ENDCASE *idstr* _ NULL;
     DROP(idsfail);
     (frtn):                                                        1E3L
     IF NOT fileno THEN close(idstid.stfile := 0);
     RETURN(outcome);
     (idsfail) CATCHPHRASE();                                       1E3O
        BEGIN
        CASE SIGNALTYPE OF
           = aborttype :
              TERMINATE; % to frtn %
           ENDCASE;
        CONTINUE;
        END;
     END.

  (idplname) %print a last name branch of individuals%
  PROCEDURE (stid, idstr);                                          1E4
     LOCAL count;
     LOCAL STRING work[2000];
     REF idstr;
     count _ 0;
     IF (stid := getsub(stid)) = stid THEN RETURN(count);
     LOOP
        BEGIN
        BUMP count;
        *work* _ SF(stid) SE(stid);
        idpind($work, &idstr);
        IF getftl(stid) OR inptrf := FALSE THEN EXIT LOOP;
        stid _ getsuc(stid);
        END;
     RETURN(count);
     END.

  (idpind) %format and print abreviated individual's ident entry%
  PROCEDURE (string, idstr);                                        1E5
     LOCAL STRING work[250];
     LOCAL TEXT POINTER  orgf, orge, namef, namee;
     REF idstr;
     getiid(string, &idstr, 0,0);
     getiorg(string, 0, $orgf, $orge);
     getinam(string, 0, $namef, $namee);
     *work* _ namef namee, ", Organization: ", orgf orge, ", Ident
     = ", *idstr*, EOL;
```

```
      typelit(nocawait, $work);
      RETURN;
      END.

   (idpgrporg) %format abbreviated group or organization entry%
   PROCEDURE (string, idstr);                                    1E6
      LOCAL STRING work[250];
      LOCAL TEXT POINTER namef, namee;
      REF idstr;
      getiid(string, &idstr, 0,0);
      getinam(string, 0, $namef, $namee);
      *work* _ namef namee, ", Ident = ", *idstr*, EOL;
      typelit(nocawait, $work);
      RETURN;
      END.

   (namesearch) %search and print routine%
   PROCEDURE (string, idstr, type, fileno);                      1E7
      %This procedure accepts a string containing a last name in
      namstr, and searches the identification file for entries with
      matching string -- last name, first character(s) of last name
      for individuals, or string in name...depending on type. When a
      match is found information is typed to the user, which is
      intended to identify the entry.  The number of hits is
      returned, plus if the number is greater than 0, idstr has the
      ident of the last one processed.%

      LOCAL foundsome, retvalue,  savrubabt, stid, typid, idstid, i;
      LOCAL TEXT POINTER tx1, tx2, tx3;
      LOCAL STRING work[1000], tempsr[200], uppercasestring[200];
      REF idstr, string;
      %Open id file%
         idstid _ orgstid;
         IF NOT fileno THEN
            BEGIN
            idstid.stfile _ open(0, jflname($"identfile"));
            INVOKE(clsem,klsrtn);
            END
         ELSE idstid.stfile _ fileno;
      savrubabt _ rubabt := FALSE;
      retvalue _ 0;
      CASE type OF
         = iname: %individual's last names%
            BEGIN
            FOR i _ 1 UP UNTIL > string.L DO %remove leading spaces%
               IF *string*[i] # SP THEN
                  BEGIN
                  IF i NOT= 1 THEN
                     *string* _ *string*[i TO string.L];
                  EXIT;
                  END;
            FOR i _ string.L DOWN UNTIL <= 0 DO %remove trailing
            spaces%
               IF *string*[i] # SP THEN
                  BEGIN
                  IF i NOT= string.L THEN
               *string* _ *string*[1 TO i];
            EXIT;
            END;
      *tempsr* _ "", *string*, "";
      FOR i _ 1 UP UNTIL > tempsr.L DO %convert spaces to ""%
         IF *tempsr*[i] = SP THEN *tempsr*[i] _ "";
      IF (stid _ namelook(idstid, $tempsr)) = endfil OR NOT
      (FIND SF(stid) [")] $SP "LAST NAME") THEN
         BEGIN
         typelit(nocawait, $"
         None
         ");
         EXIT CASE;
         END;
      *work* _
         "
         The following individuals with last name ",
         *string*,
         " are already defined
         ";
      typelit(nocawait, $work);
      retvalue _ idplname(stid, &idstr);
      END;
   = idchr:
      BEGIN
      IF (stid _ namelook(idstid, $""individuals"")) = endfil
      OR (stid := getsub(stid)) = stid THEN EXIT CASE;
      *work* _
         "
         The following individuals with last name beginning
         with the letter(s)  ",
         *string*,
         " are already defined
         ";
      typelit(nocawait, $work);
      FOR i _ 1 UP UNTIL > string.L DO %remove leading spaces%
         IF *string*[i] # SP THEN
            BEGIN
            IF i NOT= 1 THEN
               *string* _ *string*[i TO string.L];
            EXIT;
            END;
      *tempsr* _ *string*;
      FOR i _ 1 UP UNTIL > tempsr.L DO %convert spaces to ""%
         IF *tempsr*[i] = SP THEN *tempsr*[i] _ "";
      astruc($tempsr); %force upper case%
      LOOP
         BEGIN
         IF FIND SF(stid) $SP "( $SP  "" ^tx1 [")] ^tx2 _tx2
         $SP "LAST NAME" THEN
            BEGIN
            *uppercasestring* _ + tx1 tx2;
            IF FIND SF(*uppercasestring*) *tempsr* THEN
               retvalue _ retvalue + idplname(stid, &idstr);
            END;
         IF getftl(stid)  OR inptrf := FALSE THEN EXIT LOOP;
```

```
            stid _ getsuc(stid);
            END;
         IF NOT retvalue THEN
            typelit(nocawait, $"
            None
            ");
         END;
      = strinname: %scan names for lit%
         BEGIN
         *uppercasestring* _ *string*;
         astruc($uppercasestring);
         %process individuals%
            IF (stid _ namelook(idstid, $"'individuals'")) NOT=
            endfil AND (stid := getsub(stid)) NOT= stid THEN
               BEGIN
               *work* _
                  "
                  Following is a list of individuals which have
                  ",
                  *string*,
                  " in their names:
                  ";
               typelit(nocawait, $work);
               foundsome _ 0;
               LOOP
                  BEGIN
                  IF (FIND SF(stid)  [')] $SP "LAST NAME") AND
                  (stid := getsub(stid)) NOT= stid THEN
                     LOOP
                        BEGIN
                        *work* _ SF(stid) SE(stid);
                        getifnf($work, $tempsr,0,0); %first name
                        first%
                        astruc($tempsr);
                        IF FIND SF(*tempsr*) [*uppercasestring*]
                        THEN
                           BEGIN
                           idpind($work, &idstr);
                           BUMP foundsome;
                           END;
                        IF inptrf := FALSE THEN EXIT LOOP 2;
                        IF getftl(stid) THEN
                           BEGIN
                           stid _ getsuc(stid);
                           EXIT LOOP;
                           END;
                        stid _ getsuc(stid);
                        END;
                  IF getftl(stid) OR inptrf := FALSE THEN EXIT
                  LOOP;
                  stid _ getsuc(stid);
                  END;
               IF NOT foundsome THEN
                  typelit(nocawait, $"
                  None
                  ");
            retvalue _ retvalue + foundsome;
            END;
      %process groups%
         IF (stid _ namelook(idstid, $"'groups'")) NOT= endfil
         AND (stid := getsub(stid)) NOT= stid THEN
            BEGIN
            *work* _
               "
               Following is a list of groups which have  ",
               *string*,
               " in their names:
               ";
            typelit(nocawait, $work);
            foundsome _ 0;
            LOOP
               BEGIN
               *work* _ SF(stid) SE(stid);
               getinam($work, $tempsr,0,0);
               astruc($tempsr);
               IF FIND SF(*tempsr*) [*uppercasestring*] THEN
                  BEGIN
                  idpgrporg($work, &idstr);
                  BUMP foundsome;
                  END;
               IF getftl(stid) OR inptrf := FALSE THEN EXIT
               LOOP;
               stid _ getsuc(stid);
               END;
            IF NOT foundsome THEN
               typelit(nocawait, $"
               None
               ");
            retvalue _ retvalue + foundsome;
            END;
      %process organizations%
         IF (stid _ namelook(idstid, $"'organizations'")) NOT=
         endfil AND (stid := getsub(stid)) NOT= stid THEN
            BEGIN
            *work* _
               "
               Following is a list of organizations which have
                ",
               *string*,
               " in their names:
               ";
            typelit(nocawait, $work);
            foundsome _ 0;
            LOOP
               BEGIN
               *work* _ SF(stid) SE(stid);
               getinam($work, $tempsr,0,0);
               astruc($tempsr);
               IF FIND SF(*tempsr*) [*uppercasestring*] THEN
                  BEGIN
                  idpgrporg($work, &idstr);
                  BUMP foundsome;
                  END;
               IF getftl(stid)  OR inptrf := FALSE THEN EXIT
               LOOP;
               stid _ getsuc(stid);
               END;
            IF NOT foundsome THEN
               typelit(nocawait, $"
               None
               ");
            retvalue _ retvalue + foundsome;
            END;
         END;
      ENDCASE;
   rubabt _ savrubabt;
   (klsrtn):                                                   1E7L
   IF NOT fileno THEN close(idstid.stfile := 0);
   RETURN(retvalue);
   (clsem) CATCHPHRASE();                                      1E7C
      BEGIN
      CASE SIGNALTYPE OF
         = aborttype :
            TERMINATE; % to klsrtn %
         ENDCASE;
      CONTINUE;
      END;
   END.

(getpointer) %convert possible string address to a-string, leave
stids alone%
PROCEDURE (ptr);                                                1E8
   %handle stid, a-string, or address of string%
   IF NOT ptr.LH THEN ptr _ asrref(ptr);
   RETURN(ptr);
   END.

(orgrptst) %test if a group or organization entry%
PROCEDURE (string, dstptr);                                     1E9
   %This routine looks at string, and returns true or false to
   indicate whether it is an organization.  In addition, if the
   second argument is non-zero, it assumes that this is the
   address of a t-pointer and updates the pointer to point to the
   beginning of the membership list.  If the membership list is
   not present, then the t-pointer will contain endchr.%
   %-------------------%
   %fixed to also work with a-string or stid%
   LOCAL strptr;
   REF string, dstptr;
   strptr _ getpointer(&string);
   IF FIND SF(strptr) [')] $(SP/TAB) THEN
      BEGIN %statement name%
      FIND "Expand" $(SP/TAB); %skip "Expand" if it occurs%
      IF FIND ("Group"/"Organization") [EOL] THEN
         BEGIN %organization/group present%
         IF &dstptr THEN %set dstptr to beginning of membership
         list%
            FIND $(SP/TAB) ^dstptr;
            RETURN(TRUE);
            END;
         END;
      RETURN(FALSE);
      END.

FINISH


% SENDMAIL routines shared via INCLUDE with IDENTIFICATION %             2
   % load and nail down idents.master file.....%
      (loaidfil) PROCEDURE;  %load the Master Ident File Read Only%  2A1
         LOCAL fileno;
         fileno _ open(0, jflname($"identfile"));
         [flntadr(fileno)].flnoclos _ TRUE;
         RETURN(fileno);
         END.

   % check idents against ident-file....%
      (cknlsid) %validate ident for nls user%
      PROCEDURE (identsr, infostr, fileno);                          2B1
         %This procedure is used to validate idents for use in entering
         nls.  Currently it assumes that any ident that isn't a group
         or organization is legal%
         %-----------------%
         IF NOT infostr THEN
            IF NOT oldid(identsr, fileno) THEN RETURN(FALSE)
            ELSE NULL
         ELSE
            IF NOT ckident(identsr, infostr, fileno) THEN
            RETURN(FALSE);
         IF orgrptst(identsr, 0) THEN RETURN(FALSE);
         RETURN(TRUE);
         END.

      (ckidlist) %check if list of idents are already being used%
      PROCEDURE (idlist, badidlist, fileno);                         2B2
         %this routine assumes an identlist in IDLIST and returns with
         IDLIST containing a list of valid (in use) idents and appends
         to BADIDLIST a list of those idents which are not now in use.%
         LOCAL startstid, idstid, stid;
         LOCAL TEXT POINTER z1, z2, z3, z4, c1, c2;
         LOCAL STRING
            identsr[50], work[50], comment[150], special[10],
            origlist[500];
         REF idlist, badidlist;
         %-----------------%
         IF idlist.L = empty THEN RETURN;
         IF badidlist.L THEN *badidlist* _ *badidlist*, SP;
         %If fileno is non-zero, it is assumed to be the file number of
         the ident file...otherwise, the ident file is opened%
            idstid _ orgstid;
            IF NOT fileno THEN
               BEGIN
               INVOKE (clsfile);
               idstid.stfile _ open(0, jflname($"identfile"));
```

```
          END
        ELSE
          idstid.stfile _ fileno;
    *origlist* _ *idlist*;
    *idlist* _ NULL;
    FIND SF(*origlist*) ^c2;
    IF (startstid _ namelook(idstid, $"'usedids'")) NOT = endfil
    AND (startstid := getsub(startstid)) NCT= startstid THEN
       LOOP  %process each ident in origidlist%
          BEGIN
          FIND c2 $(SP/',) ^z3 $1('^/'&) ^z4 ^z1 $(LD/'-) ^z2 $SP
          ('( ^c1 _c1 [')] ^c2 / ^c1 ^c2);
          *identsr* _ +z1 z2;
          IF NOT identsr.L THEN EXIT; %no more idents%
          *comment* _ c1 c2;
          *special* _ z3 z4;
          *work* _',, *identsr*, ',;
          stid _ startstid;
          LOOP  %scan the list of assigned idents to find this
          one%
             BEGIN
             FIND SF(stid) ^z1;
             WHILE ( FIND z1 [*identsr*] ^z1 ) DO
                IF FIND < [',] > *work* THEN
                   BEGIN
                   *idlist* _ *idlist*, *special*, *identsr*,
                   *comment*, SP;
                   EXIT LOOP 2;
                   END;
             IF getftl(stid) THEN
                BEGIN
                *badidlist* _ *badidlist*, *special*, *identsr*,
                *comment*, SP;
                EXIT LOOP;
                END;
             stid _ getsuc(stid);
             END;
          END;
    IF NOT fileno THEN close(idstid.stfile := 0);
    IF idlist.L THEN BUMPDOWN idlist.L;
    IF badidlist.L THEN BUMPDOWN badidlist.L;
    RETURN;
    END.

  (oldid) %check if an ident is already being used%
  PROCEDURE (identsr, fileno);                                      2B3
    %this routine returns true if the ident passed is aready in
    use, false otherwise -- Much faster than ckident%
    LOCAL idstid, stid;
    LOCAL TEXT POINTER z1;
    LOCAL STRING work[200], ident[100];
    REF identsr;
    %If fileno is non-zero, it is assumed to be the file number of
    the ident file...otherwise, the ident file is opened%
       IF identsr.L = empty THEN RETURN(FALSE);
       idstid _ orgstid;
```

```
            IF NOT fileno THEN
               BEGIN
               INVOKE (clsfile);
               idstid.stfile _ open(0, jflname($"identfile"));
               END
            ELSE
               idstid.stfile _ fileno;
         *ident* _ *identsr*;
         astruc(&identsr);
         *work* _',, *identsr*, ',;
         IF (stid _ namelook(idstid, $"'usedids'")) NOT = endfil AND
         (stid := getsub(stid)) NOT= stid THEN
            LOOP
               BEGIN
               FIND SF(stid) ^z1;
               WHILE ( FIND z1 [*identsr*] ^z1 ) DO
                  IF FIND < [',] > *work* THEN
                     BEGIN
                     IF NOT fileno THEN close(idstid.stfile := 0);
                     *identsr* _ *ident*;
                     RETURN(TRUE);
                     END;
               IF getftl(stid) THEN EXIT LOOP;
               stid _ getsuc(stid);
               END;
         IF NOT fileno THEN close(idstid.stfile := 0);
         *identsr* _ *ident*;
         RETURN(FALSE);
         (clsfile) CATCHPHRASE ;                                         2B3N
            BEGIN
            CASE SIGNALTYPE OF = aborttype:
               IF idstid.stfile THEN close(idstid.stfile := 0);
               ENDCASE;
            CONTINUE;
            END;
         END.
   % Retrieve logical fields from ident-file entry....%
      (luser) %get LOGICAL USER-NAME (for NLS delivery) for an ident
      entry%
      PROCEDURE (entrystr, fldstr, ptr1, ptr2);                          2C1
         LOCAL TEXT POINTER lptr1, lptr2;
         LOCAL retval;
         REF entrystr;
         retval _ FALSE;
         IF NOT
            (FIND SF(*entrystr*) ["User:"]
               $NP  ^lptr1 [';] < CH $NP > ^lptr2) THEN
                  getilname(&entrystr, fldstr, ptr1, ptr2)
         ELSE
            BEGIN
            stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
            retval _ TRUE;
            END;
         RETURN(retval);
         END.
```

```
(lgetsubcoll) %get LOGICAL SUBCOLLECTIONS for an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                          2C2
   LOCAL TEXT POINTER lptr1, lptr2;
   LOCAL retval;
   LOCAL STRING tempsr[30];
   REF entrystr;
   IF NOT (retval _ getisubcol(&entrystr, fldstr, ptr1, ptr2))
   THEN
      IF orgrptst(&entrystr, 0) THEN getiid(&entrystr, fldstr,
      ptr1, ptr2)
      ELSE
         BEGIN
         getiorg(&entrystr, $tempsr, 0, 0);
         IF lhostn # nsahost THEN IF *tempsr* # "SRI-ARC" THEN
         *tempsr* _ "NIC";
         FIND SF(*tempsr*) ^lptr1 SE(*tempsr*) > ^lptr2;
         stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
         END;
   RETURN(retval);
   END.

(laddress) %get LOGICAL U.S. MAIL ADDRESS for an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2, fileno);                  2C3
   % Fileno is the file number of an open file in which the
   address ident is checked; it is passed to ckident. If zero,
   the identfile is used and opened.  Entrystr is the address of
   a string containing the text of the ident entry whose mailing
   address is sought; fldstr is the address of a strig into which
   the mailing address is placed.  ptr1 and ptr2 are either the
   address of two text pointers which will delimit the  mailing
   address in the field string passed; they may be zero.  The
   procedure returns false if the address is not found. %
   %Get the mailing address, even if you have to go to another
   ident for it -- New Version by WLB 5 July 1972; modified by
   HGL 18-DEC-73%
   LOCAL retval;
   LOCAL TEXT POINTER tpf, tpe;
   LOCAL STRING savedid[30];
   REF fldstr, entrystr, ptr1, ptr2;
   IF NOT &fldstr THEN RETURN (FALSE);
   getiadd(&entrystr, &fldstr, &ptr1, &ptr2);
   IF NOT FIND SF(*fldstr*) [NP] THEN
      BEGIN %See if it is an ident%
      % save away this ident:  the entrystr will be smashed and
      then restored using it.  This is done to avoid using space.
       It is slow and could be avoided by using another string. %
      getiid (&entrystr, $savedid, 0, 0);
      IF ckident(&fldstr, &entrystr, fileno) THEN
         BEGIN % entrystr now has the info for the ident (a group
         or person) which has the address. %
         retval _ laddress(&entrystr, &fldstr, &ptr1, &ptr2,
         fileno);
         getinam (&entrystr, 0, $tpf, $tpe);
         IF orgtst (&entrystr, 0) THEN % Must have organization
         name at beginning %
            *fldstr* _ tpf tpe, EOL, *fldstr*
            ELSE *fldstr* _ "c/o ", tpf tpe, EOL, *fldstr*;
            % Now restore original string. %
            IF ckident ( $savedid, &entrystr, fileno ) AND retval
            THEN
               BEGIN
               IF &ptr1 THEN FIND SF(*fldstr*) ^ptr1;
               IF &ptr2 THEN FIND SE(*fldstr*) ^ptr2;
               RETURN(TRUE);
               END
            ELSE RETURN( FALSE);
            END;
         IF NOT ckident ( $savedid, &entrystr, fileno ) THEN RETURN
         (FALSE); % Return FALSE if we could not restore the
         original information %
         END;
      IF &ptr1 THEN FIND SF(*fldstr*) ^ptr1;
      IF &ptr2 THEN FIND SE(*fldstr*) ^ptr2;
      RETURN (TRUE);
      END.

   (lmemlist) %get LOGICAL MEMBERSHIP for a group/organization
   ident%
   PROCEDURE (entrystr, outlist, explist, noexplist, errorlist,
   idfile);                                                       2C4
      % Gets the logical membership list of a group or organization
      -- i.e., expands membership list (recursively) until only
      individual idents are present in list.  This routine is
      supposed to correctly handle expand/noexpand issues and also
      will not loop in the event of circular references -- e.g.,
      GROUPA is a member of GROUPB is a member of GROUPC is a member
      of GROUPA -- isn't that a cute feature!
         ENTRYSTR is the name of a string containing the text of an
         identfile entry (as copied from the identfile) -- the name,
         coordinator, and memlist of the group/organization being
         referenced will be extracted from this string.
            NOTE: this must be a legitimate L10 STRING, as it will
            be used for working space by lmemlist -- the original
            contents will be restored before lmemlist returns,
            however.
         OUTLIST is the name of a string in which the logically
         expanded memlist will be returned -- it must be large
         enough to hold the requested results!
            In the output list each individual's ident is followed
            by a parenthetical expression containing a list of his
            "capacities" seperated by spaces.
            A person's capacity is the ident of a group or
            organization in whose memlist his ident appears.
               E.g., if GROUPC is a member of GROUPB, GROUPB is a
               member of GROUPA, and JOEBLOW is a member of GROUPC,
               then JOEBLOW's capacity is GROUPC.
            If an individual has more than one capacity in a group,
            then all will be listed in the parenthetical expression
            following his ident in the output list -- i.e., his
            ident itself will appear only once.
               E.g., if GROUPC is a member of GROUPB, GROUPB is a
               member of GROUPA, and JOEBLOW is a member of GROUPA
            and GROUPC, then JOEBLOW's part of OUTLIST would be:
               JOEBLOW(GROUPA GROUPC)
         The first ident listed in OUTLIST will be that of the
         coordinator of the group/organization whose memlist is
         being expanded.
      EXPLIST is the name of a string in which will be returned a
      list of idents (each followed by a space) of all
      groups/organizations which have been expanded in the course
      of producing OUTLIST  -- i.e., if an ident appears as a
      capacity on OUTLIST it will appear as an entry on EXPLIST.
      NOEXPLIST is the name of a string in which will be returned
      a list of idents (each followed by a space) of all
      groups/organizations which have been encountered but not
      expanded in the course of producing OUTLIST -- i.e., idents
      which were found preceeded by '& and idents for no-expand
      groups/organizations which were not found preceeded by '^.
         NOTE that no group/organization which is found on
         NOEXPLIST will be found as a capacity on OUTLIST -- if a
         group/organization appears in both an expand and a
         no-expand context, the expand context takes precedent.
      ERRORLIST is the name of a string in which will be returned
      a list of idents (each followed by a space) which were
      encountered in the course of producing OUTLIST and which
      could not be found in the identfile.
      IDFILE is the file number of the identfile, if open when
      lmemlist was called, or zero.
      %
   LOCAL lmlflag, expchr;
   LOCAL TEXT POINTER lp1, lp2, lp3, lp4, inp, expp, noexpp;
   LOCAL STRING inlist[500], inid[20], capacity[20], idstr[20];
   REF entrystr, outlist, explist, noexplist, errorlist;
   IF NOT orgrptst (&entrystr,0) THEN RETURN(FALSE);
   % Initialize Group Scan %
      getiid (&entrystr, 0, $lp1, $lp2); *inid* _ +lp1 lp2;
      *explist* _ *inid*, SP;
      FIND SF(*explist*) ^expp;
      *outlist* _ NULL;
      *noexplist* _ NULL;
      *errorlist* _ NULL;
      lmlflag _ TRUE;
   LOOP BEGIN
      IF NOT FIND expp > ^lp1 [SP] ^expp ^lp2 _lp2 THEN EXIT
      LOOP;
      *capacity* _ lp1 lp2;
      IF lmlflag
         THEN lmlflag _ FALSE
         ELSE IF NOT ckident ($capacity, &entrystr, idfile)
            THEN BEGIN
               *errorlist* _ *errorlist*, *capacity*, SP;
               REPEAT LOOP;
               END;
      getimem (&entrystr, 0, $lp1, $lp2); *inlist* _ +lp1 lp2;
      geticord (&entrystr, 0, $lp1, $lp2); *idstr* _ +lp1 lp2;
      WHILE ( FIND SE(*inlist*) '; ) DO BUMPDOWN inlist.L;
      IF idstr.L # 0 AND FIND SF(*inlist*) [ *idstr* ^lp2
      -(LD/'-) < lp2 1$(LD/'-) $SP ^lp1 > $SP *idstr* ]
```

```
         THEN *inlist* _ *idstr*, SP, SF(*inlist*) lp1, lp2
         SE(*inlist*)
         ELSE *inlist* _ *idstr*, SP, *inlist*;
         % INLIST now has mem list with coordinator's id at front
         %
      FIND SF(*inlist*) ^inp;
      LOOP BEGIN
         FIND inp > $( $NP ^inp '( [')] );
         expchr _ IF FIND inp '^ $NP ^inp THEN '^
            ELSE IF FIND inp '& $NP ^inp THEN '&
            ELSE 0;
         IF NOT FIND inp ^lp1 L $(LD/'-) ^lp2 ^inp THEN EXIT
         LOOP;
         *idstr* _ lp1 lp2;
         IF NOT ckident ($idstr, &entrystr, idfile)
            THEN BEGIN
               *errorlist* _ *errorlist*, *idstr*, '(,
               *capacity*, '), SP;
               REPEAT LOOP;
               END;
         IF orgrptst (&entrystr, 0)
            THEN BEGIN
               IF FIND SF(*explist*) ( *idstr* SP / [ SP *idstr*
               SP ] )
                  THEN NULL
                  ELSE CASE expchr OF
                     = '^ : BEGIN
                        *explist* _ *explist*, *idstr*, SP;
                        END;
                     = '& : IF NOT FIND SF(*noexplist*) ( *idstr*
                     SP / [ SP *idstr* SP ] )
                        THEN *noexplist* _ *noexplist*, *idstr*,
                        SP;
                     ENDCASE IF expdtst (&entrystr, 0)
                        THEN REPEAT CASE ('^)
                        ELSE REPEAT CASE ('&);
               END
            ELSE IF FIND SF(*outlist*) ( *idstr* '( / [ SP
            *idstr* '( ] ) [')] ^lp1 _lp1
               THEN *outlist* _
                  SF(*outlist*) lp1, SP, *capacity*, lp1
                  SE(*outlist*)
               ELSE *outlist* _
                  *outlist*, *idstr*, '(, *capacity*, '), SP;
         END; %LOOP%
      END; %LOOP%
   FIND SF(*noexplist*) ^noexpp;
   lmlflag _ FALSE;
   WHILE ( FIND noexpp ^lp1 [SP] ^lp2 _lp2 ^noexpp) DO BEGIN
      *capacity* _ lp1 lp2;
      IF FIND SF(*explist*) ( *capacity* SP / [ SP *capacity* SP
      ] )
         THEN BEGIN
            FIND noexpp < ^lp1 SP 1$(LD/'-) ^noexpp;
            *noexplist* _ SF(*noexplist*) noexpp, lp1
            SE(*noexpp*);
```

```
               END;
            END;
         ckident ($inid, &entrystr, idfile); % Restore entrystr %
         RETURN (TRUE);
         END.
% retrieve fields from entry in ident-file....%
      (getiadd) %get MAIL ADDRESS field from an ident entry%
      PROCEDURE (entrystr, fldstr, ptr1, ptr2);                     2D1
         LOCAL TEXT POINTER lptr1, lptr2;
         REF entrystr;
         IF orgrptst(&entrystr, 0) THEN
            FIND SF(*entrystr*) 4[EOL] $SP ^lptr1
         ELSE
            FIND SF(*entrystr*) 2[EOL] ^lptr1;
         IF NOT (FIND lptr1 [EOL EOL] < $NP ^lptr2 >) THEN
            FIND lptr1 > ^lptr2;
         stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
         RETURN (TRUE);
         END.
      (getilname) %get LAST NAME field from an individual ident entry%
```

SHERWOOD

SHERWOOD

```
    PP@@@@@@@@@DD@@@      (      """""yqppp              b""""tp   p     tp0
   %Get last name%
   LOCAL TEXT POINTER lptr1, lptr2;
   REF entrystr;
   getinam(&entrystr, 0, $lptr1, $lptr2);
   FIND lptr1 > [',] ^lptr2_lptr2;
   stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
   RETURN (TRUE);
   END.

(getiexp) %get EXPAND field from an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                              2D3
   LOCAL TEXT POINTER lptr1, lptr2;
   REF entrystr;
   FIND SF(*entrystr*) [')] $NP ^lptr1 ("Expand"/) ^lptr2;
   stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
   RETURN (TRUE);
   END.

(getimem) %get MEMBERSHIP field from a group/organization ident
entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                              2D4
   LOCAL TEXT POINTER lptr1, lptr2;
   REF entrystr;
   IF NOT orgrptst(&entrystr, 0)
   OR NOT
      (FIND SF(*entrystr*) [EOL] $(SP/TAB) ^lptr1 PT [EOL] < $NP
      ^lptr2 >) THEN
         FIND SF(*entrystr*) [EOL] ^lptr1 ^lptr2;
   stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
   RETURN (TRUE);
   END.

(getiprevmem) %get MEMBERSHIPPREVIOUS field from an ident entry%
PROCEDURE (entrystr, fldstr, ptr1, ptr2);                              2D5
```

```
      LOCAL TEXT POINTER lp1, lp2;
      LOCAL retval;
      REF entrystr;
      retval _ FALSE;
      IF NOT FIND SF(*entrystr*) ["MembershipPrevious:"] $NP ^lp1
      [";] < CH $NP > ^lp2 THEN
         getiend (&entrystr, $lp1, $lp2)
      ELSE retval _ TRUE;
      stptset (fldstr, ptr1, ptr2, $lp1, $lp2);
      RETURN (retval);
      END.

   (geticord) %get COORDINATOR field from a group/organization ident
   entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        2D6
      LOCAL TEXT POINTER lptr1, lptr2;
      REF entrystr;
      IF NOT orgrptst(&entrystr, 0)
      OR NOT
         (FIND SF(*entrystr*) 3[EOL] $(SP/TAB) ^lptr1 PT
            [EOL] <$NP> ^lptr2 ) THEN
               FIND SF(*entrystr*) 3[EOL] ^lptr1 ^lptr2;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN (TRUE);
      END.

   (getityp) %get TYPE-OF-ORGANIZATION field from an organization
   ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        2D7
      LOCAL TEXT POINTER lptr1, lptr2;
      REF entrystr;
      IF NOT orgrptst(&entrystr, 0)
      OR NOT
         (FIND SF(*entrystr*) ["Type:"] $NP ^lptr1 [";] < CH $NP >
         ^lptr2 ) THEN
            getiend(&entrystr, $lptr1, $lptr2);
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN (TRUE);
      END.

   (getiverify) %get VERIFIED-BY-NIC-PERSONNEL field from an ident
   entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        2D8
      LOCAL TEXT POINTER lptr1, lptr2;
      REF entrystr;
      IF NOT
         (FIND SF(*entrystr*) ["Verified"/"Unverified"] ^lptr2
            < [EOL] > $NP ^lptr1) THEN
               getiend(&entrystr, $lptr1, $lptr2);
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN (TRUE);
      END.

   (getiphone) %get PHONE field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                        2D9
      LOCAL TEXT POINTER lptr1, lptr2;
      LOCAL retval;
      REF entrystr;
      retval _ FALSE;
      IF NOT
         (FIND SF(*entrystr*) ["Phone:"] $NP ^lptr1 [';] < CH $NP >
            ^lptr2) THEN
               getiend(&entrystr, $lptr1, $lptr2)
      ELSE retval _ TRUE;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN(retval);
      END.

   (getifunction) %get FUNCTION field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                    2D10
      LOCAL TEXT POINTER lptr1, lptr2;
      LOCAL retval;
      REF entrystr;
      retval _ FALSE;
      IF NOT
         (FIND SF(*entrystr*) ["Function:"] $NP ^lptr1 [';] < CH $NP
         >
            ^lptr2) THEN
               getiend(&entrystr, $lptr1, $lptr2)
      ELSE retval _ TRUE;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN(retval);
      END.

   (getisorg) %get SECONDARY ORGANIZATION field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                    2D11
      LOCAL TEXT POINTER lptr1, lptr2;
      LOCAL retval;
      REF entrystr;
      retval _ FALSE;
      IF NOT
         (FIND SF(*entrystr*) ["Secondary organization:"] $NP ^lptr1
         [';] < CH $NP > ^lptr2 ) THEN
            getiend(&entrystr, $lptr1, $lptr2)
      ELSE retval _ TRUE;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN(retval);
      END.

   (getisubcol) %get SUBCOLLECTIONS field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                    2D12
      LOCAL TEXT POINTER lptr1, lptr2;
      LOCAL retval;
      REF entrystr;
      retval _ FALSE;
      IF NOT
         (FIND SF(*entrystr*) ["Sub-Collection:"] $NP ^lptr1 [';] <
         CH $NP >
            ^lptr2 ) THEN
               getiend(&entrystr, $lptr1, $lptr2)
      ELSE retval _ TRUE;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN(retval);
      END.

   (getimcmnts) %get COMMENTS field from an ident entry%
   PROCEDURE (entrystr, fldstr, ptr1, ptr2);                    2D13
      LOCAL TEXT POINTER lptr1, lptr2;
      LOCAL retval;
      REF entrystr;
      retval _ FALSE;
      IF NOT
         (FIND SF(*entrystr*) ["Comments:"] $SP ^lptr1 [EOL] ^lptr2
         _lptr2)
            THEN FIND SE(*entrystr*) ^lptr1 ^lptr2
      ELSE retval _ TRUE;
      stptset(fldstr, ptr1, ptr2, $lptr1, $lptr2);
      RETURN(retval);
      END.

% support routines for get/set entry in ident-file....%
   (expdtst) %test expand field and set text-pointer to membership
   list%
   PROCEDURE (string, dstptr);                                   2E1
      %This routine looks at the statement indicated by srcptr, and
      returns true or false to indicate whether the expand parameter
      is set.  In addition, if the second argument is non-zero, it
      assumes that this is the address of a t-pointr and updates the
      pointer to point to the beginning of the membership list.  If
      the membership list is not present, then the t-pointer will
      contain endchr.%
      %--------------------%
      REF string, dstptr;
      IF FIND SF(*string*) '( $(LD/''/'-) ') $(SP/TAB)
         "Expand" $(SP/TAB)
         ("Group"/"Organization") $(SP/TAB) EOL
         THEN
            BEGIN %expand parameter set%
            IF &dstptr THEN %set dstptr to beginning of membership
            list%
               IF FIND ^dstptr $(SP/TAB) THEN
                  FIND ^dstptr;
            RETURN(TRUE);
            END
      ELSE RETURN(FALSE);
      END.

% test type of entry in ident-file....%
   (jgrptst) %test if a group entry%
   PROCEDURE (string, dstptr);                                   2F1
      %This routine looks at the statement indicated by srcptr, and
      returns true or false to indicate whether it contains a group
      identification.  In addition, if the second argument is
      non-zero, it assumes that this is the address of a t-pointr
      and updates the pointer to point to the beginning of the
      membership list.%
      %--------------------%
      REF string, dstptr;
      IF FIND SF(*string*) [")] $(SP/TAB) %statement name%
         (("Expand" $(SP/TAB) "Group")/"Group")
         $(SP/TAB) EOL THEN
            BEGIN %group id present%
            IF &dstptr THEN %set dstptr to beginning of membership
            list%
               IF FIND ^dstptr $(SP/TAB) THEN
                  FIND ^dstptr;
            RETURN(TRUE);
            END
      ELSE RETURN(FALSE);
      END.

   (orgtst) %test if a organization entry%
   PROCEDURE (string, dstptr);                                    2F2
      %This routine looks at the statement indicated by srcptr, and
      returns true or false to indicate whether it is an
      organization.  In addition, if the second argument is
      non-zero, it assumes that this is the address of a t-pointer
      and updates the pointer to point to the beginning of the
      membership list.  If the membership list is not present, then
      the t-pointer will contain endchr.%
      %--------------------%
      REF string, dstptr;
      IF FIND SF(*string*) [")] $(SP/TAB) %satatement name%
         (("Expand" $(SP/TAB) "Organization")/"Organization")
         $(SP/TAB) EOL  THEN
            BEGIN %organization id present%
            IF &dstptr THEN %set dstptr to beginning of membership
            list%
               IF FIND ^dstptr $(SP/TAB) THEN
                  FIND ^dstptr;
            RETURN(TRUE);
            END
      ELSE RETURN(FALSE);
      END.

% expand list of idents into list of individuals....%
   (getids) %expand list of idents into list of individuals%
   PROCEDURE (ptr, astr, infotype, idfile);                       2G1
      LOCAL expchr, gpstid;
      LOCAL TEXT POINTER idf, ide, tmpptr, srcptr, dstptr;
      LOCAL STRING idstr[20], infostr[2000];
      REF ptr, astr;
      %reads an ident from pointer ptr into astring idstr, and then
      calls ckident to get info on it.
         IF infotype = 0, then it returns all of the info in astr,
         if infotype = 1, then the name only is returned.
         Uses infostr as a work area%
      expchr _ 0;
      %first, read ident%
         LOOP
            BEGIN
            CCPOS ptr;
            IF FIND $NP ^idf "; THEN
               BEGIN
            IF NOT popids(&ptr) THEN RETURN(FALSE); %no more
            idents%
            END
         ELSE EXIT LOOP;
         END;
      IF NOT FIND idf [ NP / '; / '( ]
         ^ptr _ptr ^ide _ide THEN
            err($"Ident List Format Error");
      FIND ptr ($NP '( [')] ^ptr);
      IF FIND idf ('&/'^) ^idf  < CH > THEN expchr _ READC;
      *idstr* _ idf ide; %ident%
   %Now get info, and check ident%
      IF ckident($idstr, $infostr, idfile : gpstid) THEN %return
      something%
         BEGIN
         IF orgrptst($infostr, 0) THEN
            BEGIN
            expchr _ CASE expchr OF %expand group list or not%
               ='&: FALSE;
               ='^: TRUE;
               ENDCASE expdtst($infostr, 0); %take default from
               ident record%
            IF expchr THEN
               BEGIN
               getimem($infostr, 0,  $dstptr, 0);
               IF FIND dstptr -EOL %membership list present% THEN

                  BEGIN
                  pushids(&ptr);
                  dstptr _ gpstid;
                  FIND dstptr ^ptr;
                  RETURN(getids(&ptr, &astr, infotype, idfile));
                  END;
               END;
            END;
         END;
      %Now edit and append to astr %
         IF infotype = 1 THEN
            getifnf($infostr,  $infostr);
         *astr* _ *astr*, *infostr*;
   RETURN(TRUE) END.

% ident pushdown stack support....%
  % miscellaneous utility routines....%
  (intids) %initialize ident pushdown stack%
  PROCEDURE (ptr);                                        2H2
     RESET jidstk;
     IF ptr THEN pushids(ptr);
     jidsbot _ jidstk;
     RETURN;
     END.

  (popids) %pop the ident pushdown stack%
  PROCEDURE (ptr);                                        2H3
     REF ptr;
     IF jidsbot = jidstk THEN RETURN(FALSE);
```

```
      POP jidstk TO ptr;
      RETURN(TRUE);
      END.

   (pushids) %push the ident pushdown stack%
   PROCEDURE (ptr);                                                 2H4
      REF ptr;
      PUSH ptr ON jidstk;
      RETURN;
      END.

   (gbotids) %collaps ident pushdown stack%
   PROCEDURE (ptr);                                                 2H5
      REF ptr;
      IF jidsbot = jidstk THEN RETURN(FALSE);
      IF jidstk.systks=1 THEN ptr _ [$jidstk+2]
      ELSE mvbfbf($jidstk+2, &ptr, jidstk.systks);
      RETURN(TRUE);
      END.

% Retrieve logical fields from ident-file entry....%
% miscellaneous utility routines %
   (getgpids) %given an identlist, return list of only the group
   idents%
   PROCEDURE (ptr, astr, idfnum);                                   2J1
      %Read identlist identified by ptr, and return all group idents
      referenced in the string astr, separated by spaces%
      LOCAL TEXT POINTER z1, z2, z3;
      LOCAL idfile;
      LOCAL STRING idstr[20], infostr[2000];
      REF ptr, astr;
      IF NOT idfnum THEN
         BEGIN
         idfile _ 0;
         INVOKE(clsfile);
         idfile _ open(0, jflname($"Identfile"));
         END
      ELSE idfile _ idfnum;
      FIND ptr ^z1;
      WHILE (FIND z1 $NP ('^/'&/) ^z2 -'; [ NP / '; / '(] < CH > ^z3
       ( $NP '( ['')] / ) ^z1 ) DO
         BEGIN
         *idstr* _ z2 z3;
         IF NOT ckident($idstr, $infostr, idfile) THEN
         err($"Identification System Error");
         IF orgrptst($infostr, 0) THEN *astr* _ *astr*, SP, *idstr*;
         END;
      IF NOT idfnum THEN close(idfile := 0);
      RETURN;
      (clsfile) CATCHPHRASE ;                                       2J1L
         BEGIN
         CASE SIGNALTYPE OF = aborttype:
             sigclose(idfile := 0);
           ENDCASE;
         CONTINUE;
         END;
      END.
   (makgid) %generate an ident from group name string%
   PROCEDURE (namestr, idstr);                                      2J2

      LOCAL count;
      LOCAL TEXT POINTER tptr1, tptr2;
      REF namestr, idstr;
      count _ empty + 1;
      *idstr* _ *namestr*[count];
      FIND SF(*namestr*) ^tptr1 ^tptr2;
      WHILE (count _ count+1) <= idstr.M DO
         IF (FIND tptr1 [SP] [L] ^tptr1 _tptr1 ^ tptr2) THEN
            *idstr* _ *idstr*, tptr1 tptr2
         ELSE EXIT LOOP;
      RETURN;
      END.

   (clnidlst) %clean ident list -- of comments, expansion chars,
   etc.%
   PROCEDURE (inlist, %appends to->% outlist);                      2J3
      %----------%
      LOCAL STRING ident [40];
      LOCAL TEXT POINTER l1, l2, lp1, lp2, lp3;
      REF inlist, outlist;
      %----------%
      FIND SF(*outlist*) ^l1;
      IF outlist.L AND *outlist* [outlist.L] # SP THEN
         *outlist* _ *outlist*, SP;
      FIND SF(*inlist*) ^lp3;
      WHILE (FIND lp3 > $(SP/',) ('&/'^/) ^lp1 1$(LD/'-) ^lp2
      ($(SP/',) '( [')] /) ^lp3) DO
         BEGIN
         *ident* _ lp1 lp2;
         FIND SE(*outlist*) ^l2;
         IF NOT ckipmem ($ident, $l1, $l2) THEN
            *outlist* _ *outlist*, *ident*, SP;
         END;
      IF *outlist* [outlist.L] = SP THEN BUMP DOWN outlist.L;
      RETURN;
      END.

   (delcmidents) %delete idents common to both ident lists%
   PROCEDURE (lhlist, lhdelete, rhlist, rhdelete);                  2J4
      %----------%
      LOCAL STRING lhident [40], rhident [40];
      LOCAL TEXT POINTER lh1, lh2, lht, rh1, rh2, rht;
      REF lhlist, rhlist;
      %----------%
      FIND SE(*lhlist*) (';/) ^lh1;
      WHILE (FIND lh1 < ^lht $(SP/',) ^lh2 1$(LD/'-) ^lh1) DO
         BEGIN
         *lhident* _ lh1 lh2;
         FIND SE(*rhlist*) (';/) ^rh1;
         WHILE (FIND rh1 < ^rht $(SP/',) ^rh2 1$(LD/'-) ^rh1) DO
            BEGIN
            *rhident* _ rh1 rh2;
            IF *lhident* = *rhident* THEN
               BEGIN
               IF lhdelete THEN ST lh1 lht _ NULL;
               IF rhdelete THEN ST rh1 rht _ NULL;
               EXIT LOOP;
               END;
            END;
         END;
      IF lhdelete AND FIND SE(*lhlist*) (';/) ^lh2 1$(SP/',) ^lh1
      THEN
         ST lh1 lh2 _ NULL;
      IF rhdelete AND FIND SE(*rhlist*) (';/) ^rh2 1$(SP/',) ^rh1
      THEN
         ST rh1 rh2 _ NULL;
      RETURN;
      END.
```

%FORMAT OF IDENTFILE

```
   The identfile is an NLS file <IDENTFILE>IDENTS.MASTER of the
   following format
      origin statement
         ('usedids')                                                 3A1A
            number,ident,ident,ident,...,ident,
               where there are number idents in the statement.  when
               number reaches 200, a new statement is created.
            .
            .
            .
         ('individuals')                                             3A1B
            ('lastname') LAST NAME                                   3A1B1
               (ident) OrganizationIdent
               Lastname, Firstname MiddleInitial. (nickname), Jr.
               Address

                                                                     3A1B1A
                  where all of the idents have the same lastname (the
                  name of the head of the branch).
               .
               .
               .
            .
            .
            .
         ('groups')                                                  3A1C
            (ident) Expand Group
            membershiplist
            groupname
            cordinatorIdent
            Address

                                                                     3A1C1
            .
            .
            .
         ('organizations')                                           3A1D
            (ident) Organization
```

```
        membershiplist
        groupname
        cordinatorIdent
        Address
                                                      3A1D1
        .
        .
        .
%
```