JNL DEL

```
<NLS>JNLDEL.NLS;158, 2-JAN-72 23:43 WSD ;
FILE jnldel % L10 <rel-NLS>jnldel %
%Declarations %
   REF tda;
(hcdex)PROCEDURE;
   LOCAL distcount, collcount, niccount, char, collections,
   distribution, nic;
   LOCAL STRING passw[5], jnum[5], tempsr[5];
   %Hard copy distribution execute%
   %Parse start-up command%
      %Password%
         *passw* ← NULL;
         echoff();
         crlf();
         prompt($"Password:");
         UNTIL passw.L = 3 DO
            IF (char ← inpcuc()) = CD THEN
               GOTO STATE
            ELSE *passw* ← *passw*, char;
         IF *passw* # "JPD" THEN
            BEGIN
            tqmarks();
            GOTO STATE;
            END;
      % Get operators ident. %
         crlf();
         prompt($"Operator: ");
         *opstr* ← NULL;
         rdident($opstr,0);
         IF curchr # CA THEN getctc();
   %Initialise Parameters%
      %Global stids and  file variables%
         docjfn ← hdrjfn ← docstrt ← lastfnum ← lptused ← lptjfn ←
         idfile ← prntfg ← 0;
      %Control paameters%
         distribution ← collections ← TRUE;
         nic ← FALSE;
      %set up subsystem name for statistics%
         r1 ← getsbn($"HCJDEL");
         !JSYS setnm;
   %Now see which outpu processor and printer file we are to use%
      IF jdebug THEN
         BEGIN
         %Debugging--use normal output processor and print file xlpt%
         *ptstr* ← "XLPT.TXT";
         *oprocn* ← *opname*;
         END
      ELSE
         BEGIN % real system..normal output processor and lpt%
         *ptstr* ← "LPT:";
         *oprocn* ← *opname*;
         END;
   %Type a warning message if this is the experimental system%
      IF jdebug THEN
         BEGIN
         crlf();
```

```
      typeas($"Experimental Journal System");
      END;
%Now update hcdistfile from distfile%
   IF updhcdist() = 0 THEN
      BEGIN
      dismes(1, $"Distribution File Empty");
      jdquit();
      END;
%Now parse control%
   STATE ← jdlstate, spec;
   thearald('$);
   IF tnumber($jnum) > 0 THEN
      BEGIN
      IF curchr # CA THEN error();
      crlf();
      printnum($jnum, distribution, collections, nic); %
      number--output specific document%
      END
   ELSE CASE IF curchr IN ['a, 'z] THEN curchr-40B ELSE curchr OF
      ='A:
         BEGIN
         echo($"All");
         getctc();
         collections ← distribution ← nic ← TRUE;
         END;
      ='C: %Collection Copies%
         BEGIN
         echo($"Collection Copies ");
         collections ← answer();
         END;
      ='D: %Distribution%
         BEGIN
         echo($"Distribution Copies ");
         distribution ← answer();
         END;
      ='G: %Go%
         BEGIN
         echo($"Go");
         getctc();
         crlf();
         IF NOT (distribution OR collections OR nic) THEN err($"No
         Parameters Set");
         rubabt ← 1;
         printall(distribution, collections, nic);
         END;
      ='N: %Nic%
         BEGIN
         echo($"NIC Copies ");
         nic ← answer();
         END;
      ='O: %Output Processor Version%
         BEGIN
         echo($"Output Processor Version ");
         CASE inpcuc() OF
            ='E, ='X:
               BEGIN
```

```
            echo($"Experimental");
            getctc();
            *oprocn* ← *xopname*;
            END
         = 'N:
            BEGIN
            echo($"Normal");
            getctc();
            *oprocn* ← *opname*;
            END
         ='?:
            help($"Experimental",
               $"Normal",
               0);
         ENDCASE tqmarks();
   END;
='P:
   BEGIN
   echo($"Printer File Name ");
   *lit* ← NULL;
   txtlit($lit);
   *ptstr* ← *lit*;
   END;
='Q:
   BEGIN
   echo($"Quit ");
   getctc();
   jdquit();
   END;
='R: %reset parms%
   BEGIN
   echo($"Reset Parameters");
   getctc();
   distribution ← collections ← nic ← FALSE;
   END;
='S:
   BEGIN
   echo($"Status");
   getctc();
   crlf();
   distcount ← hcstatus(:collcount, niccount);
   %type out distribution count%
      crlf();
      typeas($"Distribution:  ");
      *tempsr* ← STRING(distcount);
      typeas($tempsr);
   %Type out Collection count%
      crlf();
      typeas($"Collection:  ");
      *tempsr* ← STRING(collcount);
      typeas($tempsr);
   %Type out Nic count%
      crlf();
      typeas($"Nic:  ");
      *tempsr* ← STRING(niccount);
      typeas($tempsr);
```

```
            %type out total%
               crlf();
               typeas($"Total ");
               *tempsr* ← STRING(distcount+collcount+niccount);
               typeas($tempsr);
            END;
         ='?:
            help($"Document Number",
               $"All",
               $"Collection Copies ",
               $"Distribuiton Copies ",
               $"Go",
               $"NIC Copies ",
               $"Output Processor Version ",
               $"Printer File Name ",
               $"Quit",
               $"Reset Parameters",
               $"Status",
               0);
      ENDCASE tqmarks();
      GOTO STATE;
   END.
(printall)PROC(distribution, collections, nic);
   %This procedure prints all of the indicated copies of all documents
   in the hcdistfile%
   LOCAL hcdstid, stid;
   hcdstid ← 0;
   ON SIGNAL ELSE
      BEGIN
      sigclose(hcdstid.stfile := 0);
      lptclose();
      END;
   hcdstid ← openlock(0, jflname($"hcdistfile"));
   pfilnum ← 1;
   stid ← getsub(hcdstid);
   WHILE stid # hcdstid DO
      BEGIN
      printdoc(stid:= getsuc(stid), distribution, collections, nic);
      BUMP pfilnum;
      END;
   close(hcdstid.stfile := 0);
   lptclose();
   RETURN;
   END.
(printnum)PROC(numstr, distribution, collections, nic);
   %Print the document indicated by numstr if there is an entry for it
   in the hcdistfile%
   LOCAL hcdstid, stid;
   LOCAL STRING tempsr[10];
   REF numstr;
   hcdstid ← 0;
   ON SIGNAL ELSE sigclose(hcdstid.stfile := 0);
   *tempsr* ← 'J, *numstr*;
   hcdstid ← openlock(0, jflname($"hcdistfile"));
   IF (stid  ← namelook(hcdstid, $tempsr))= endfil THEN err($"No Such
   document in distfile");
```

```
    pfilnum ← 1;
    printdoc(stid, distribution, collections, nic);
    lptclose();
    close(hcdstid.stfile := 0);
    RETURN;
    END.

(printdoc)PROC(hcdstid, distribution, collections, nic);
    %Print out the various flavors of the document identified by distfile
    entry hcdstid.
       Flags indicate which copies to print%
    LOCAL dwstid, stid;
    LOCAL TEXT POINTER z1, z2;
    LOCAL STRING jnum[5];
    dwstid ← 0;
    prntfg ← TRUE;
    ON SIGNAL ELSE
       BEGIN
       close(dwstid.stfile := 0);
       IF sysgnl # -4 THEN %Not called to rubout%
          BEGIN
          dismes(1, sysmsg);
          RETURN;
          END;
       END;
    %get document number into jnum, and type it to the operator%
       FIND SF(hcdstid) ↑z1;
       *jnum* ← *[getjnm($z1)]*;
       crlf();
       typeas($"Document #");
       typeas($jnum);
       crlf();
    %set up document for printing%
       %Get author names and addresses for address page%
          getdauth(hcdstid, $lit2);
       %Set up te dwork file%
          dwstid ← getdwork(hcdstid, $lit2);
    %Do distribution copies first%
       IF distribution THEN
          BEGIN
          stid ← getsub(hcdstid);
          WHILE stid # hcdstid DO
             pdistcopy(dwstid, stid := getsuc(stid), $jnum);
          END;
    %Do collection copies%
       IF collections AND NOT FIND SF(hcdstid) [EOL "Secondary
       Distribution Copy"] THEN
          BEGIN
          collcopy(dwstid, hcdstid, $"Master", $jnum);
          collcopy(dwstid, hcdstid, $"Access", $jnum);
          collcopy(dwstid, hcdstid, $"Engelbart", $jnum);
          END;
    %Do NIC copies%
       IF nic AND FIND SF(hcdstid) ["Sub-Collections: "] ↑z1 [';] ↑z2
       BETWEEN z1 z2(["NIC"]) THEN
          BEGIN
```

```
        collcopy(dwstid, hcdstid, $"NIC", $jnum);
        nicsitecopies(dwstid, hcdstid);
        END;
    close(dwstid.stfile := 0);
    RETURN;
    END.

(getdauth)PROC(stid, authsr);
    %Set up astr authsr to have the list of authors of document indicated
    by distfile entry stid.  Append address of first author to end of
    string%
    LOCAL idfno;
    LOCAL STRING tempsr[30], tempadsr[150];
    LOCAL TEXT POINTER z1, z2;
    REF authsr;
    idfno ← 0;
    ON SIGNAL ELSE
        BEGIN
        sigclose(idfno := 0);
        IF sysgnl # -4 THEN SIGNAL(1);
        END;
    IF NOT FIND SF(stid) ["Author(s): "] ['/] ↑z1 ('&/'↑/) ↑z2 THEN
        SIGNAL(1, $"Bad Distfile Entry");
    *authsr* ← NULL;
    idfno ← open(0, jflname($"identfile"));
    %Get the names of the authors%
        intids(0);
        WHILE getids($z1, &authsr, 1, idfno) DO
            *authsr* ← *authsr*, EOL;
    %Now get the addresss of the frst author%
        IF (authsr.L = 0) OR NOT FIND z2 L $LD ↑z1 THEN err($"Illegal
        Author Field");
        *tempsr* ← z2 z1;
        IF NOT ckident($tempsr, $xlit, idfno) THEN err($"Illegal Author");
        close(idfno := 0);
    %Now extract address and put it all together%
        laddress($xlit, $tempadsr, 0, 0);
        *authsr* ← *authsr*, *tempadsr*;
    RETURN;
    END.

(getdwork)PROC(hcdstid, authsr);
    %Open and set up dwork file for printing.
        Return stid of dwork file if ok, zero if not.
        %
    REF authsr;
    LOCAL dwstid, docstid;
    LOCAL TEXT POINTER z1, z2, z3;
    dwstid ← docstid ← 0;
    ON SIGNAL ELSE
        BEGIN
        close(dwstid.stfile := 0);
        close(docstid.stfile := 0);
        END;
    %Open dwork, and set up the origin statement%
        dwstid ← openlock(0, jflname($"distwork"));
```

```
      enablaccess(dwstid.stfile, jrnlaccess);
      cer(dwstid.stfile);
      FIND SF(hcdstid) [')] ↑z1 ( $NP '( [')] / ) ↑z2 (([/"Master Copy
      Printed"/
         "Access Copy Printed"/
         "Engelbart Copy Printed"]) < [EOL] > /
         SE(z1) > ) ↑z3;
      ST dwstid ← *authsr*, ".LMS=20;
      .GCR=3; To: .LMS=24;

      .PEL;
      .LMS = 0;", SF(hcdstid) z1, z2 z3;
   %Now set up the rest of the file%
      IF docstid ← jdelloaddoc(hcdstid) THEN
         BEGIN
         ccp(dwstid, getsub(docstid), sucdir);
         close(docstid.stfile := 0);
         END;
   RETURN(dwstid);
   END.

(jdelloaddoc)PROC(stid);
   %assume that stid points to a distfile entry, and get the branch
   containing the coresponding document, returning stid of document or
   zero if it is hard copy%
   LOCAL docstid;
   LOCAL TEXT POINTER z1;
   LOCAL STRING user[20], filename[50], statname[30], vsp[15];
   docstid ← 0;
   ON SIGNAL ELSE
      BEGIN
      close(docstid.stfile := 0);
      SIGNAL(1);
      END;
   IF FIND SF(stid) [')] ↑z1  ["Hard Copy--Location"] THEN RETURN(0);
   %Now parse link---z1 points to spaces before it%
      lnkspec(1, $z1, $user, $filename, $statname, $vsp);
      *filename* ← *filename*, ".NLS";
   %Now open file and find proper branch%
      docstid.stpsid ← origin;
      docstid.stfile ← open(0, $filename);
      IF *statname*[1] IN ['0, '9] THEN RETURN(docstid); %a terrible way
      of checking if it is a document or message---change after backlog
      is gone %
      makeptr(docstid, $z1);
      specreg($statname, name, $z1);
      IF z1 = endfile THEN
         BEGIN
         *lit* ← *filename*, " Branch ", *statname*, " Not Found";
         err($lit);
         END;
      RETURN(z1);
   END.

(pdistcopy)PROC(dwstid, adrstid, jnumber);
   %Print a copy of the documen indicated by dwstid for the peson
```

```
   addressed by adrstid if it should be done, and mark entry adrstid as
   having been printed%
   LOCAL STRING adrstr[200];
   LOCAL TEXT POINTER z1, z2, z3, z4;
   REF jnumber;
   CCPOS SF(adrstid);
   xtrnam($adrstr, $swork, '(, ') );
   ckident($adrstr, $lit2, 0);
   IF phcopy(adrstid, $lit2, 0) THEN %A copy should be printed for this
   user%
      BEGIN
      %Get his name and address%
         getinam($lit2, 0, $z1, $z2);
         laddress($lit2, $adrstr, 0, 0);
         *adrstr* ← z1 z2, EOL, *adrstr*, ".LMS=60;.GCR=5;", *jnumber*,
         ".LMS=0;.GCR;";
         %Now look for possible comments%
            IF FIND SF(adrstid) ("Note: " ↑z1 ";;" ↑z2  %New format%
               /(".GCR=5;.GCR=5;" ↑z1 EOL EOL ↑z2) ←2z2 %Old Format%
               THEN *adrstr* ← *adrstr*, ".GCR=4;.GCR=5;", z1 z2, EOL;
      %Now print it%
         printit(dwstid, $adrstr, pfilnum);
      %Now tag statement as having been printed%
         tagjdel(adrstid, $"Hard Copy");
      END;
   RETURN;
   END.
(tagjdel)PROC(stid, astr);
   REF astr;
   %Tag the statement stid with string + "Printed By " OPERATOR"%
   ST stid ← SF(stid) SE(stid), EOL, *astr*, " Printed by ", *opstr*;
   RETURN;
   END.
(collcopy)PROC(dwstid, hcdstid, astr, jnum);
   %Print a 'Collection Copy' For collection identified by astr if it
   has not already been done.  Tag statement hcdstid accordingly%
   LOCAL STRING tempsr[100];
   REF jnum, astr;
   *tempsr* ← *astr*, " Copy Printed";
   IF NOT FIND SF(hcdstid) (*tempsr*) THEN %It has not been printed%
      BEGIN
      *tempsr* ← *astr*, " Copy.LMS=60;.GCR=5;", *jnum*, ".LMS=0;.GCR;";
      printit(dwstid, $tempsr, pfilnum);
      *tempsr* ←  *astr*, " Copy";
      tagjdel(hcdstid, $tempsr);
      END;
   RETURN;
   END.

(lptclose)PROC;
   %Close out all of the lpt machinery%
   IF lptjfn THEN sysclose(lptjfn := 0);
   IF docjfn THEN sysclose(docjfn := 0);
   IF hdrjfn THEN sysclose(hdrjfn := 0);
   %Global stids and  file variables%
      docstrt ← lastfnum ← lptused ← idfile ← 0;
```

```
   RETURN;
   END.
(nicsitecopies)PROC(dwstid, hcdstid);
   RETURN;
   END.

(updhcdist)PROC;
   %Update DISTFILE from TDISTFILE, and return TRUE if there is anything
   in the distribution file%
   LOCAL ldiststid, hcdstid, retval, stid, stidd;
   LOCAL STRING adsr[45];
   %Initialise LOCALS%
      hcdstid ← 0;
      hcdstid.stpsid ← origin;
      ldiststid ← hcdstid;
   %Set Signal to close files%
      ON SIGNAL
         ELSE
            BEGIN
            sigclose(hcdstid.stfile);
            sigclose(ldiststid.stfile);
            END;
   ldiststid ← openlock(0, jflname($"distfile"));
   IF (getsub(ldiststid)).stpsid = origin THEN
      BEGIN
      close(ldistfile.stfile);
      RETURN(TRUE); %Assume that there will always be something in the
      dist file%
      END;
   hcdstid ← openlock(0,jflname($"hcdistfile"));
   enablaccess(hcdstid.stfile, jrnlaccess);
   enablaccess(ldiststid.stfile, jrnlaccess);
   cmp(stid←getail(getsub(hcdstid)), getsub(ldiststid), sucdir);
   closeu(ldiststid.stfile := 0);
   *auxlit* ← NULL;
   UNTIL (stid ← getsuc(stid)).stpsid = origin DO BEGIN
      stidd ← getsub (stid);
      UNTIL stidd = stid DO BEGIN
         CCPOS SF(stidd);
         xtrnam ($adsr, $swork, -1);
         IF adsr.L = 0 OR FIND SF(*auxlit*) [*adsr*]
            THEN cdb (stidd:= getsuc (stidd))
            ELSE BEGIN
               stidd ← getsuc(stidd);
               *auxlit* ← *auxlit*, SP, *adsr*;
               END;
         END;
      END;
   retval ← (getsub(hcdstid)).stpsid # origin;
   closeu(hcdstid.stfile := 0); %update file%
   RETURN(retval);
   END.
(hcstatus)PROC;
   LOCAL hcdstid, stid1, stid2, distcount, collcount, niccount, count,
   idjfn;
   LOCAL TEXT POINTER z1, z2;
```

```
    %return numbet of expedited printouts to be done, no. of normal docs%
    hcdstid ← 0;
    ON SIGNAL ELSE
       BEGIN
       sigclose(hcdstid.stfile := 0);
       sigclose(idjfn := 0);
       END;
    hcdstid.stpsid ← origin;
    hcdstid.stfile ← open(0, jflname($"hcdistfile"));
    idjfn ← open(0, jflname($"Identfile"));
    stid1 ← getsub(hcdstid);
    distcount ← collcount ← niccount ← 0;
    WHILE stid1 # hcdstid DO
       BEGIN
       stid2 ←  getsub(stid1);
       WHILE stid2 # stid1 DO
          BEGIN
          distcount ← distcount + phcopy(stid2, 0, idjfn);
          stid2 ← getsuc(stid2);
          END;
       collcount ← collcount +
          (NOT FIND SF(stid1) /"Master Copy Printed"/) +
          (NOT FIND SF(stid1) /"Access Copy Printed"/) +
          (NOT FIND SF(stid1) /"Engelbart Copy Printed"/);
       niccount ← niccount + ((FIND SF(stid1) /"Sub-Collections: "/ ↑z1
       /';/ ↑z2 BETWEEN z1 z2(/"NIC"/)) AND NOT FIND SF(stid1) /"NIC Copy
       Printed"/);
       stid1 ← getsuc(stid1);
       END;
    close(idjfn := 0);
    close(hcdstid.stfile := 0);
    RETURN(distcount, collcount, niccount);
    END.

(jdquit)PROCEDURE;
    %Print out hcdistfile if printing has been done, and garbage collect
    distribution file%
    LOCAL hcdstid;
    hcdstid ← 0;
    ON SIGNAL ELSE sigclose(hcdstid.stfile := 0);
    IF prntfg  THEN
       BEGIN
       prntfg ← FALSE;
       % print out distfile%
          outptr(IF jdebug THEN $"<Duvall>hcdistfile" ELSE
          $"<Journal>Hcdistfile");
       %garbage collect hcdistfile%
          hcdstid ← openlock(0, jflname($"hcdistfile"));
          gdistf(hcdstid := 0);
        END;
    %Set subsystem name back to regular nls%
       r1 ← nlssbn;
       !JSYS setnm;
    reset();
    END.
(printit)PROCEDURE(stid, astr, fnum);
```

```
   LOCAL cntr, newdoc;
   LOCAL TEXT POINTER z1, z2, z3;
   REF astr;
   %Replace current name statement if astr # 0%
      IF &astr # 0 THEN
         BEGIN
         CCPOS SF(stid);
         IF NOT FIND [".LMS=24;
         "] ↑z1 ["
         .PEL;"] [EOL] ↑ z3 < [EOL]  [EOL] ↑z2 >  THEN
            SIGNAL(2, $"Illegal Header in Printit");
         ST stid ← SF(stid) z1,*astr*,z2 z3, ".PSW=0;.HLT;", z3
         SE(stid);
         crlf();
         typeas(&astr); %let operator see what is happenning%
         END
      ELSE
         BEGIN %Garden variety document%
         lastfnum ← 0;
         poutput(stid, $docjfn, $docwfn);
         opnlpt();
         copylpt(TRUE, docjfn, 0, -1);
         clslpt();
         RETURN
         END;

   %Output header%
      poutput(stid, $hdrjfn, $hdrwfn);
   %If new file, get rest of document%
      IF (lastfnum := fnum) # fnum THEN
         BEGIN %New document%
         IF NOT FIND SF(stid) [".PSW=0;.HLT;"] ↑z2 < ["P."] ↑z1 >  THEN
            SIGNAL(2, $"Bad print file");
         ST z1 z2 ← NULL; %delete IRS and HALT%
         %output entire file%
            poutput(stid, $docjfn, $docwfn);
         %Now get start of document from header file%
            r1 ← hdrjfn;
            IF NOT SKIP !JSYS sizef THEN
               err($"Print File Error");
            docstrt ← r2-1;
         END;
   %Now copy to lpt:%
      opnlpt();
      copylpt(TRUE, hdrjfn, 0, -1); %header%
      copylpt(FALSE, docjfn, docstrt, -1); %document%
      clslpt();
   RETURN;
   END.
(poutput)PROC(stid, jfn, jfnnam);
   LOCAL opjfn;
   REF jfn, jfnnam;
   %Close file if it is open%
      IF jfn # 0 THEN  sysclose(jfn := 0);
   %now call processor%
      tda.dacsp ← stid;
```

```
      cod (jflname(&jfnnam), &tda, opprdv);
   %Now re-open file%
      jfn ← sgtjfn(getgtjflg(write, 0, oldvrsn), jflname(&jfnnam),
      $lit);
      IF NOT sysopen(jfn, readwrite, chrtyp, $lit) THEN
         BEGIN
         %May have failed for timing reasons%
         r1 ← 1000;
         !JSYS dismS;
         IF NOT sysopen(jfn, readwrite, chrtyp, $lit) THEN
            err($lit);
         END;
   RETURN END.

(copylpt)PROC(resflg, jfn, startb, stopb);
   LOCAL count;
   %First set file pointer to start%
      r1 ← jfn; r2 ← startb; IF NOT SKIP !JSYS sfptr THEN err($"I/O JSYS
      error");
      %Now set eof%
         IF stopb = -1 THEN
            BEGIN
            %get size of file%
            r1 ← jfn; IF NOT SKIP !JSYS sizef THEN err($"I/O JSYS
            error");
            stopb ← r2;
            END
         ELSE
            BEGIN
            %set size of file%
            r1.rh ← jfn; r1.lh ← 12B; r2 ← -1; r3 ← stopb;
            !JSYS chfdb;
            END;
      %Send out restore if necessary%
         %output processor paginates for us now%
      %Get count and output%
         count ← stopb - startb;
         WHILE count > 0 DO
            BEGIN
            %get string from input%
               count ← count - 2560;
               r3 ← -(IF count < 0 THEN 2560+count ELSE 2560);
               r1 ← jfn; r2 ← chbptr(0)+$rsvb9;
               !JSYS sin; %read string from input file%
            %copy to lpt%
               r1 ← lptjfn; r2 ← chbptr(0) + $rsvb9;
               r3 ← -(IF count < 0 THEN 2560+count ELSE 2560);
               !JSYS sout;
            END;
      RETURN
      END.

(opnlpt)PROC;
   LOCAL cntr;
   IF lptjfn # 0 THEN BEGIN
      BUMP lptused;
```

```
      RETURN;
      END;
   lptjfn ← lptused ← 0;
   IF (lptjfn ← sgtjfn(getgtjflg(write, 0, 0), $ptstr, $lit)) = 0 THEN
   err($lit);
   FOR cntr ← 0 UP 1 UNTIL > 90 DO
      IF sysopen(lptjfn, write, lpttype, $lit) THEN RETURN
      ELSE pause(20000);
   lptjfn ← 0;
   err($"printer busy too long");
   END.

(clslpt)PROC;
   IF lptused < 10 THEN RETURN;
   lptused ← 0;
   sysclose(lptjfn := 0);
   RETURN END.
(oldist)PROC;
   %Does the on-line distribution of JOurnal Documents%
   LOCAL fileno, fl, dirnm, stid, stidl, tstid, idstid, pcap, wmesfg,
   adrstid, hcdstid;
   LOCAL TEXT POINTER z1 , z2, z3;
   LOCAL STRING fnsr[45], tinit[15], tempsr[50], filenm[50], stat[30],
   spcvw[15];
   REF fl;
   %Set up subsy name%
      r1 ← getsbn($"OLJDEL");
      !JSYS setnm;
   %Enable access to Journal Files%
      enablaccess(0, jrnlaccess);
   %Set up signal and initialise parms%
      diststid ← hcdstid ← ctlstid ← wmesfg ← 0;
      ON SIGNAL ELSE
         BEGIN
         sigclose(diststid.stfile := 0);
         sigclose(hcdstid.stfile := 0);
         sigclose(idstid.stfile := 0);
         close(jrnlstid.stfile := 0);
         closeu(ctlstid.stfile := 0);
         END;
      %Update distribution file%
         IF updhcdist() = 0 THEN
            BEGIN
            %Distribution file empty%
            r1 ← nlssbn;
            !JSYS setnm;
            RETURN;
            END;
      %Now open distribution file%
         hcdstid ← getsub(openlock(0, jflname($"hcdistfile")));
   %Initialise the done string (auxlit)%
      *auxlit* ← NULL;
   %Open id file%
      idstid ← orgstid;
      idstid.stfile ← open(0, jflname($"identfile"));
   %Now look for people to send it to%
```

```
LOOP
   BEGIN
   IF hcdstid.stpsid = origin THEN EXIT;
   ON SIGNAL ELSE
      BEGIN
      hcdstid ← getsuc(hcdstid);
      close(jrnlstid.stfile := 0);
      closeu(ctlstid.stfile := 0);
      REPEAT LOOP;
      END;
   adrstid ← getsub(hcdstid);
   LOOP
      BEGIN
      IF adrstid = hcdstid THEN EXIT;
      CCPOS SF(adrstid);
      IF NOT FIND /"Author Copy"/ THEN
         BEGIN  %Try to distribute it%
         *fnsr* ← NULL;
         xtrnam($fnsr, $swork, -1);
         IF fnsr.L = 0
            OR (FIND SF(*auxlit*) /*fnsr*/)
            OR NOT ckident($fnsr, $xlit, idstid.stfile)
            OR NOT donline(adrstid, $xlit, idstid.stfile) THEN
            GOTO nodo;
         *auxlit* ← *auxlit*, SP, *fnsr*; %mark him as being
         tried%
         *tinit* ← *fnsr*; %save off initials%
         %Now see if we can open his control file%
            luser($xlit, $tempsr, 0, 0);
            IF NOT (ctlstid ← opnctfile($fnsr, $tempsr)) THEN GOTO
            nodo;
            %By here, ctl file is open%
            %Check for Journal Branch%
               IF (tstid ← namelook(ctlstid, $"Journal"))= endfil
               THEN
                  BEGIN %No Journal branch...make one%
                  *tempsr* ← "(Journal) Journal Documents (most
                  recent first)";
                  ctlstid ← cis(ctlstid, $tempsr, down);
                  END
               ELSE ctlstid ← tstid;
         %Now start to give him the documents%
            stid ← adrstid;
            LOOP
               BEGIN
               stidl ← getup(stid); % head of branch%
               %Set up link/location pointers%
                  CCPOS SF(stidl);
                  IF FIND /"Hard Copy--Location: "/ ↑z1 /';/ ↑z2
                  THEN GOTO uselnk
                  ELSE FIND /')/ /'(/ ↑z1 ←z1 /')/ ↑z2; %Pointers
                  to link%
                  IF FIND BETWEEN z1 z2(/"JRNL"/) THEN %message%
                     BEGIN
                     lnkspec(1, $z1, $tempsr, $filenm, $stat,
                     $spcvw);
```

```
                        *filenm* ← *filenm*, ".NLS";
                        jrnlstid ← openlock(0, $filenm);
                        FIND SF(jrnlstid) ↑z3;
                        specreg($stat, name, $z3);
                        IF z3 = endfil THEN
                           BEGIN %sumpin slipped%
                           close(jrnlstid.stfile := 0);
                           GOTO uselnk;
                           END;
                        tstid ← ccs(ctlstid, z3, down); %Copy over
                        header%
                        z1 ← getsub(z3);
                        FIND SF(z1) ↑z1 SE(z1) ↑z2; %message%
                        olhed(TRUE, tstid, $z1, $z2); %format
                        statement%
                        close(jrnlstid.stfile := 0);
                        END
                     ELSE
                        BEGIN %Put a link into his file%
                        (uselnk):
                        tstid ← ccs(ctlstid, z1, down);
                        olhed(FALSE, tstid, $z1, $z2);
                        END;
                  %Now fix up hcdistfile %
                     ST stid ← SF(stid) SE(stid), " Distributed
                     On-Line ";
                  %Now proceed to next document%
                     LOOP
                        BEGIN
                        z1 ← stid;
                        lookup($z1, $tinit, seqname);
                        IF z1 = endfil THEN EXIT 2; %done%
                        stid ← z1;
                        CCPOS SF(z1);
                        IF donline(stid, $xlit, idstid.stfile)  THEN
                        EXIT; %Do this one%
                        END; %of search loop%
                  END; %of main dstribution loop%
            %Now close his control file%
               pcap ← enablw();
               closeu(ctlstid.stfile := 0);
               IF pcap # -1 THEN disablw(pcap);
            END; %of this guy%
         (nodo): adrstid ← getsuc(adrstid);
      END; %Of this document%
      hcdstid ← getsuc(hcdstid);
   END;
%Now wrap it up%
   close(hcdstid.stfile := 0);
   close(idstid.stfile := 0);
   close(jrnlstid.stfile := 0);
   closeu(ctlstid.stfile := 0);
   r1 ← nlssbn;
   !JSYS setnm;
RETURN
END.
```

```
(enablw)PROC;
   LOCAL pcap;
   r1 ← 4B5;
   !JSYS rpcap;
   pcap ← r3;
   r3 ← r2;
   IF NOT r2 .A 6B5 THEN RETURN(-1);
   r1 ← 4B5;
   !JSYS epcap;
   RETURN(pcap)
   END.

(disablw)PROC(pcap);
   %Disable wheel status, and restore status to pcap%
   r1 ← 4B5;
   !JSYS rpcap;
   r3 ← pcap;
   r1 ← 4B5;
   !JSYS epcap;
   RETURN(pcap)
   END.

(olhed)PROC(mflag, stid, l1, l2);
   %Format stid to onmline delivery header format%
   %l1, l2 delimit location of document or message if mflag is true%
   LOCAL TEXT POINTER z1;
   REF l1, l2;
   CCPOS SF(stid);
   FIND ↑z1;
   *lit* ← *[hjournal($z1)]*,
      EOL, *[jtitle($z1)]*,
      EOL, *[hcopyl($z1)]*,
      *[olloc(mflag, &l1, &l2)]*;
   ST z1 ← *lit*;
   RETURN
   END.

(hjournal)PROC(z1);
   LOCAL TEXT POINTER z2 , z3, z4, z5;
   REF z1;
   CCPOS z1;
   IF FIND [".HJOURNAL="] ['"']  ↑z2 ['"'] ↑z3 ←z3 THEN
      BEGIN
      IF FIND BETWEEN z2 z3([".SPLIT;"] ↑z4 < ['.] ↑z5 >) THEN
         *lit2* ← z4 z3, " (", z2 z5, ')
      ELSE *lit2* ← z2 z3
      END
   ELSE *lit2* ← NULL;
   RETURN($lit2)
   END.

(jtitle)PROC(z1);
   LOCAL TEXT POINTER z2 , z3;
   REF z1;
   CCPOS z1;
```

```
   IF FIND ["Title: "] ['"] ↑z2 ['"] ↑z3 ←z3 THEN
      *lit2* ← z2 z3
   ELSE
      *lit2* ← NULL;
   RETURN($lit2)
   END.

(hcopy1)PROC(z1);
   REF z1;
   CCPOS z1;
   IF FIND ["Hard Copy--Location: "] THEN
      *lit2* ← "Hard Copy Only--"
   ELSE
      *lit2* ← NULL;
   RETURN($lit2)
   END.
(olloc)PROC(mflag, z1, z2);
   REF z1, z2;
   IF mflag THEN
      *lit2* ← "Message: "
   ELSE
      *lit2* ← "Location: ";
   *lit2* ← *lit2*,  z1 z2;
   RETURN($lit2)
   END.

(gdistf)PROC(hcdstid);
   %Garbage collect hard copy distribution file%
   LOCAL stid, stid1, idjfn;
   LOCAL TEXT POINTER z1,z2;
   LOCAL STRING tempsr[50];
   %Set up  subsy name%
      r1 ← getsbn($"GCOLDF"); !JSYS setnm;
   idjfn ← 0;
   ON SIGNAL ELSE
      BEGIN
      sigclose(idjfn := 0);
      sigclose(hcdstid.stfile := 0);
      END;
   %open hcdist file%
      IF NOT hcdstid.stfile THEN hcdstid ← openlock(0,
      jflname($"hcdistfile"));
      idjfn ← open(0, jflname($"identfile"));
   %loop through file looking for stuff to delete%
      stid ← getsub(hcdstid);
      WHILE stid # hcdstid DO
         BEGIN
         stid1 ← getsub(stid);
         WHILE stid1 # stid DO
            BEGIN
            CCPOS SF(stid1);
            xtrnam($tempsr, $swork, -1);
            ckident($tempsr, $xlit, idjfn);
            IF NOT (phcopy(stid1, $xlit, idjfn) OR donline(stid1, $xlit,
            idjfn)) THEN
               cds(stid1 := getsuc(stid1))
```

```
            ELSE stid1 ← getsuc(stid1);
            END;
         IF getsub(stid) = stid THEN
            BEGIN %all copies printed%
            CCPOS SF(stid);
            IF (FIND ↑z1 /"Master Copy Printed"/)
               AND (FIND z1 /"Access Copy Printed"/)
               AND (FIND z1 /"NIC Copy Printed"/)
               AND (FIND z1 /"Engelbart Copy Printed"/)
               THEN cds(stid := getsuc(stid))
            ELSE stid ← getsuc(stid);
            END
         ELSE stid ← getsuc(stid);
         END;
   %Update and exit%
      closeu(hcdstid.stfile := 0);
      close(idjfn);
      r1 ← getsbn($"HCJDEL");
      !JSYS setnm;
      RETURN
      END.

(phcopy)PROC(stid, namstr, idjfn);
   %Return true if hard copy needs to be printed for statement
   identified by stid.
      namstr contains 0 or id record for user, idjfn is 0 or ident file
      number%
   LOCAL TEXT POINTER z1, z2;
   LOCAL STRING idntsr/15/;
   REF namstr;
   IF FIND SF(stid) /"Hard Copy Printed"/ THEN RETURN(FALSE); %Already
   done%
   IF FIND SF(stid) /"Author Copy"/ THEN RETURN(TRUE); %All author
   copies go out hard copy%
   IF &namstr = 0 THEN
      BEGIN
      CCPOS SF(stid);
      xtrnam($idntsr, $swork, -1);
      ckident($idntsr, $lit2, idjfn);
      &namstr ← $lit2;
      END;
   RETURN((ldelivery(asrref(&namstr))).delhc);
   END.

(donline)PROC(stid, namstr, idjfn);
   %Return true if on line distribution should be done for statement
   identified by stid.
      namstr contains 0 or id record for user, idjfn is 0 or ident file
      number%
   LOCAL TEXT POINTER z1, z2;
   LOCAL STRING idntsr/15/;
   REF namstr;
   IF FIND SF(stid) /"Author Copy"/"Distributed On-Line"/ THEN
   RETURN(FALSE);
   IF &namstr = 0 THEN
      BEGIN
```

```
        CCPOS SF(stid);
        xtrnam($idntsr, $swork, -1);
        ckident($idntsr, $lit2, idjfn);
        &namstr ← $lit2;
        END;
    RETURN((ldelivery(asrref(&namstr))).delol);
    END.

FINISH jnldel
```

JOCTL

```
<NLS>JOCTL.NLS;51, 11-NOV-71 12:50 JDH ; ["INPUT"];

FILE joctl % L10 to <rel-nls>joctl %
%Declarations%
   DECLARE litv = 5, filev = 6, ncopyv = 7, jmaxsmt = 10000,
      name = 1, contnt = 3;
(jcontrol)PROC;
   %This procedure is the main control section of the Journal Submode%
   LOCAL interflag, mesflg, stid, char, sabtfg, linkflg;
   LOCAL TEXT POINTER t2, linkpl;
   LOCAL STRING linkdir[20], jnumber[5], tempsr[5], templsr[50];
   jntrint();
   sabtfg ← jsavaccess ← rfcflg ← linkflg ← FALSE;
   rubabt ← TRUE;
   STATE ← fjstate, spec;
   IF sabtfg THEN jreset()
   ELSE BUMP sabtfg; %Trap Command Deletes%
   ON SIGNAL
      ELSE jclosfl();
   jctlcint();
   interflag ← 0;
   mesflg ← xsubmit($jnumber); %parse SUBMIT command%
   STATE ← jstate, spec;
   ON SIGNAL
      %de-activate previous signal%
         =2: jerror(sysmsg); %openlock fail--fatal error%
         =-4: GOTO STATE;
      ELSE;
   jolock(); %Make sure everything is ok%
   thearald('&); todco('&);
   CASE IF interflag THEN interflag := interflag-1  ELSE inpcuc() OF
      =CD: GOTO STATE;
      ='A:
         BEGIN
         porecho(interflag,$"Author(s): ");
         identlist($identstr);
         setjauthor($identstr);
         END;
      ='C:
         BEGIN
         todco('C);
         CASE inpcuc() OF
            ='L:
               BEGIN
               porecho(interflag,$"lerk: ");
               *identstr* ← NULL;
               echon();
               tirdid($identstr, 0, 0);
               IF curchr # CA THEN %Identlist not terminated by CA%
                  err($"Clerk may be only one person");
               setjclerk($identstr);
               END;
            ='O:
               BEGIN
               porecho(interflag,$"omments: ");
               *lit* ← NULL;
```

```
            txtlit($lit);
            setjcomment($lit)
            END;
         ENDCASE twohelp($"Clerk", $"Comments", 0);
      END;
   ='D, =-2:
      BEGIN
      porecho(interflag,$"Distribution: ");
      identlist($identstr);
      setjdist($identstr);
      END;
   ='E: %No Expedie for now
      BEGIN
      porecho(interflag,$"Expedite ");
      setjexpedite(answer())
      END;
      %
   ='G, =-4:
      BEGIN
      porecho(interflag,$"Go?  ");
      interflag ← 0;
      IF NOT answer() THEN GOTO STATE;
      crlf();
      typeas($"Journal System In Progress");
      jouex($jnumber,mesflg,linkflg, $linkpl,$linkdir);
      END;
   ='I:
      BEGIN
      porecho(interflag,$"Interrogate");
      getctc();
      interflag ← -1;
      END;
   ='K:
      BEGIN
      porecho(interflag,$"Keywords: ");
      *lit* ← NULL;
      txtlit($lit);
      setjkeyword($lit);
      END;
   ='O:
      BEGIN
      porecho(interflag,$"Obsoletes Document(s): ");
      *templsr* ← NULL;
      WHILE tnumber($tempsr) > 0 DO
         BEGIN
         *templsr* ← *templsr*, SP, *tempsr*;
         IF curchr = CA THEN EXIT
         ELSE typech(SP);
         END;
      IF curchr # CA THEN error();
      setjobsoletes($templsr);
      END;
   ='P:
      BEGIN
      porecho(interflag,$"Place link (if successful) ?");
      linkflg ← setlinkloc(answer(), $linkpl, $linkdir);
```

```
    END;
='Q:
    BEGIN
    porecho(interflag,$"Quit");
    getctc();
    jreset();
    END;
='S, =-3:
    BEGIN
    porecho(interflag,$"S");
    CASE IF interflag THEN interflag ELSE (char ← inpcuc()) OF
        ='T, =-4:
            BEGIN
            porecho(interflag,$"tatus");
            getctc();
            jstatus(jworkstid, linkflg, $linkpl, $linkdir);
            END;
        ='U:
            BEGIN
            porecho(interflag,$"ubcollection(s): ");
            rdsubcoll($lit);
            IF lit.L # 0 THEN setjsubcol($lit, jwpl);
            END;
        ENDCASE twohelp($"Status", $"Subcollection(s):", 0);
    END;
='T, =-1:
    BEGIN
    porecho(interflag,$"Title: ");
    *lit* ← NULL;
    txtlit($lit);
    setjtitle($lit);
    END;
='U:
    BEGIN
    porecho (interflag, $"Updates Document(s): ");
    *templsr* ← NULL;
    WHILE tnumber($tempsr) > 0 DO
        BEGIN
        *templsr* ← *templsr*, SP, *tempsr*;
        IF curchr = CA THEN EXIT
        ELSE typech(SP)
        END;
    IF curchr # CA THEN error();
    setjupdates($templsr);
    END;
= '?:
    BEGIN
    interflag ← 0;
    help($"Author(s): ",
        $"Clerk",
        $"Comments",
        $"Distribution: ",
        % $"Expedite ", %
        $"Go? ",
        $"Interrogate",
        $"Keywords: ",
```

```
            $"Obsoletes Document(s): ",
            $"Place link (if successful) ?",
            $"Quit",
            $"Status",
            $"Subcollection(s): ",
            $"Title: ",
            $"Updates Document(s): ",
            0);
         END;
      ENDCASE
         BEGIN
         interflag ← 0;
         error()
         END;
   GOTO STATE;
   END.

(djcontrol)PROC;
   %This procedure is the main control section of the Journal Submode%
   LOCAL interflag, mesflg, stid, char, sabtfg, litfg, next, linkflg;
   LOCAL TEXT POINTER t2, linkpl;
   LOCAL TEXT POINTER s1, s2;
   LOCAL STRING rfcstr[100];
   LOCAL STRING jnum[5];
   LOCAL STRING linkdir[20];
   IF jdebug THEN dismes(2, $"Experimental Journal-- Do Not Use")
   ELSE
      IF jolock() THEN
         err($"Journal system not available--try again in 10 min.");
   %set up subsystem name for statistics%
      IF NOT jnlsbn THEN jnlsbn ← <AUXCOD, getsbn>($jnlnam);
      r1 ← jnlsbn;
      !JSYS setnm;
   sabtfg ← jsavaccess ← rfcflg ← linkflg ← FALSE;
   STATE ← djstate, spec;
   IF sabtfg THEN jreset()
   ELSE BUMP sabtfg; %Trap Command Deletes%
   ON SIGNAL
      ELSE jclosfl();
   jctlcint();
   interflag ← 0;
   mesflg ← dsubmit(); %parse SUBMIT command%
   STATE ← dstate, spec;
   dismes(0);
   dn($"");
   DSP(< "Journal Command Sub-mode:");
   ON SIGNAL
      %de-activate previous signal%
      ELSE;
   jolock(); %Make sure everything is ok%
   CASE IF interflag THEN interflag := interflag-1  ELSE inpcuc() OF
      ='A, =-1:
         BEGIN
         DSP(←<  "Author(s):" );
         didentlist($identstr, $"Author(s):");
         setjauthor($identstr);
```

```
        rstlit();
        END;
    ='C, =-5:
        BEGIN
        DSP(←<  "Comments:");
        *lit* ← NULL;
        litfg ← FALSE;
        INPUT (SP BUG s1 BUG s2 CA litfg ← TRUE; / TEXT lit CA);
        IF litfg THEN
            BEGIN
            tdr($s1,$s2,$s1,$s2);
            *lit* ← s1 s2;
            END;
        setjcomment($lit);
        rstlit();
        END;
    ='D, =-3:
        BEGIN
        DSP(←<  "Distribution:");
        didentlist($identstr, $"Distribution:");
        setjdist($identstr);
        rstlit();
        END;
    ='E:
        BEGIN
        DSP(←<  "Expedite?");
        setjexpedite(danswer())
        END;
    ='G, =-8:
        BEGIN
        DSP(←<  Go );
        interflag ← 0;
        IF input()#CA THEN GOTO STATE;
        dismes(1,$"Journal System in Progress");
        jouex($jnumber,mesflg,linkflg,$linkpl,$linkdir);
        END;
    ='I:
        BEGIN
        DSP(←<  "Interrogate");
        IF input()#CA THEN GOTO STATE;
        interflag ← -1;
        END;
    ='K, =-4:
        BEGIN
        DSP(←<  "Keywords");
        *lit* ← NULL;
        litfg ← FALSE;
        INPUT (SP BUG s1 BUG s2 CA litfg ← TRUE; / TEXT lit CA);
        IF litfg THEN
            BEGIN
            tdr($s1,$s2,$s1,$s2);
            *lit* ← s1 s2;
            END;
        setjkeyword($lit);
        rstlit();
        END;
```

```
    ='O:
        BEGIN
        DSP(←<  "Operator:");
        *identstr* ← NULL;
        drdident($identstr,0);
        setjclerk($identstr);
        END;
    ='P, =-6:
        BEGIN
        DSP(←<  "Place link (if successful)");
        linkflg ← setlinkloc(danswer(), $linkpl,$linkdir);
        END;
    ='Q:
        BEGIN
        DSP(←<  "Quit");
        IF input()#CA THEN GOTO STATE;
        jreset();
        END;
    ='S, =-7:
        BEGIN
        DSP(←<S);
        CASE IF interflag THEN interflag ELSE (char ← inpcuc()) OF
            ='T, =-8:
                BEGIN
                DSP(?OFF ←<  "Status");
                IF input()#CA THEN GOTO STATE;
                jstatus(jworkstid, linkflg, $linkpl, $linkdir);
                UNTIL input()=CA DO NULL;
                jolock();
                rstlit();
                END;
            ='U:
                BEGIN
                DSP(?OFF ←<  "Subcollection membership");
                *lit* ← NULL;
                CASE (char ← inpcuc()) OF
                    ='A:
                        BEGIN
                        DSP(↑OFF);
                        *lit* ← *lit*, "ARC ";
                        litdpy($lit);
                        REPEAT;
                        END;
                    ='N:
                        BEGIN
                        DSP(↑OFF);
                        *lit* ← *lit*, "NIC ";
                        litdpy($lit);
                        REPEAT;
                        END;
                    =CA: NULL;
                    =CD: GOTO STATE;
                    =SP,=',:
                        BEGIN
                        DSP(↑OFF);
                        REPEAT;
```

```
                    END;
                NOT IN ['A, 'Z]:
                    BEGIN
                    DSP(↑OFF);
                    *lit* ← *lit*,"??";
                    litdpy($lit);
                    FIND SE(*lit*) [SP/',] ↑t2;
                    *lit* ← SF(*lit*) t2;
                    litdpy($lit);
                    REPEAT;
                    END;
                ENDCASE
                    BEGIN
                    DSP(↑OFF);
                    DO
                        BEGIN
                        *lit* ← *lit*, char;
                        litdpy($lit);
                        END
                    UNTIL (char ← inpcuc()) NOT IN ['A,'Z];
                    *lit* ← *lit*, SP;
                    litdpy($lit);
                    REPEAT(char);
                    END;
            IF lit.L # 0 THEN setjsubcol($lit, jwpl);
            rstlit();
            END;
        ENDCASE
            BEGIN
            qm();
            REPEAT(inpcuc());
            END;
    END;
  ='T, =-2:
    BEGIN
    DSP(←<  "Title:");
    *lit* ← NULL;
    litfg ← FALSE;
    INPUT (SP BUG s1 BUG s2 CA litfg ← TRUE; / TEXT lit CA);
    IF litfg THEN
        BEGIN
        tdr($s1,$s2,$s1,$s2);
        *lit* ← s1 s2;
        END;
    setjtitle($lit);
    rstlit();
    END;
  ENDCASE
    BEGIN
    interflag ← 0;
    qm();
    END;
GOTO STATE;
END.

(xsubmit)PROC(number);
```

```
%Parse and execute the submit command--Return Journal number in
numbr%
LOCAL stype, mesflg, rnumstid, cnumstid, useflg, char;
LOCAL TEXT POINTER z1, z2;
LOCAL STRING tempsr[100], authstr[50];
REF number;
rnumstid ← cnumstid ← useflg ← 0;
ON SIGNAL ELSE
   BEGIN
   sigclose(rnumstid.stfile := 0);
   sigclose(cnumstid.stfile := 0);
   END;
stype ← getjdocument(:mesflg); %get id of journal dicument..returns
tyype and stid's in spec stack, and lit with literal if stype = 1lit%
crlf();
prompt($"Number: ");
echoff();
IF tnumber(&number) > 0 THEN
   BEGIN %Pre-assigned number%
   cnumstid ← openlock(0, jflname($"tcnumbers"));
   IF NOT rdcknum(1, &number, $authstr,  cnumstid :cnumstid) THEN
      BEGIN
      unlktst();
      useflg ← 1;
      END;
   echoff();
   CCPOS SF(cnumstid);
   IF FIND [" NWG/RFC "] THEN rfcflg ← 2;
   close(cnumstid.stfile := 0);
   IF rfcflg THEN
      BEGIN %fix up rfc file%
      rnumstid ← openlock(0, jflname($"rfcnumbers"));
      *tempsr* ← '#,  *number*;
      makeptr(rnumstid, $z1);
      lookup($z1, $tempsr, contnt);
      IF z1 = endfil THEN
         err($"RFC Number Error--Call NIC");
      CCPOS SF(z1);
      IF (FIND ["INUSE"]) AND NOT useflg THEN unlktst()
      ELSE ST z1 ← SF(z1) SE(z1), " INUSE ";
      CCPOS SF(z1);
      IF NOT FIND ['C/ ↑z1 1$D ↑z2 $NP ') THEN  err($"RFC Number
      Error--Call NIC");
      *rfcnum* ← z1 z2;
      close(rnumstid.stfile := 0);
      crlf();
      typeas($"RFC #");
      typeas($rfcnum);
      END;
   END
ELSE CASE (char ← curchr) OF
   =CA:
      BEGIN
      typech(char);
      echon();
      getcnum($initsr, &number, $" JOURNAL ", 0, 0);
```

```
        typeas(&number);
        *authstr* ← *initsr*; %ident of author%
        END
    ='r, ='R: %RFC Number%
        BEGIN
        echo($"RFC Number: ");
        IF tnumber($rfcnumber) > 0 THEN
           BEGIN %pre-assigned%
           typech(curchr);
           echon();
           rnumstid ← openlock(0, jflname($"rfcnumbers"));
           IF NOT racknum(1, $rfcnum, $authstr, rnumstid :rnumstid)
           THEN
              BEGIN
              unlktst();
              useflg ← 1;
              END;
           echoff();
           CCPOS SF(rnumstid);
           IF NOT FIND ['#] ↑z1 4$D ↑z2 THEN err($"RFC Number
           Error--Call NIC");
           *number* ← z1 z2;
           close(rnumstid.stfile := 0);
           %Now check and mark tcnumber file%
              cnumstid ← openlock(0, jflname($"tcnumbers"));
              IF NOT numtype(cnumstid,&number, $"PREASSIGNED"
              :cnumstid) THEN err($"RFC Number Error--Call NIC");
              IF FIND SF(cnumstid) ["INUSE"] THEN
                 BEGIN
                 IF NOT useflg THEN
                    unlktst();
                 END
              ELSE ST SF(cnumstid) ← SF(cnumstid) SE(cnumstid), " INUSE
              ";
              close(cnumstid.stfile := 0);
           rfcflg ← 2;
           crlf();
           typeas($"Catalog #");
           typeas(&number);
           END
        ELSE CASE (char ← curchr) OF
           =CA: BEGIN
              %get rfc number%
              rnumstid ← openlock(0, jflname($"rfcnumbers"));
              rnumstid ← getcnum($initsr, $rfcnum, $" NWG/RFC ", 0,
              rnumstid);
              getcnum($initsr, &number, $" NWG/RFC ", 0, 0);
              ST SF(rnumstid) ← SF(rnumstid) SE(rnumstid), EOL, '#,
              *number*;
              close(rnumstid.stfile := 0);
              BUMP rfcflg;
              typeas($rfcnum);
              crlf();
              typeas($"Catalog #");
              typeas(&number);
              END;
```

```
            ENDCASE error();
         END;
      ='?:
         help($"Pre-assigned CATNUM",
            $"CA",
            $"RFC Number ",
            0);
      ENDCASE error();
   %By here, number = number, authstr = author, stype = type, and
   b1,b2,lit contain parms depending on type%
   %Now open work file and set up header statement%
      setjwork(stype, &number, $authstr);
      IF rfcflg THEN
         BEGIN
         %If pre-assigned RFC, set up pre-entered fields of distribution
         and title%
            IF rfcflg = 2 THEN
               BEGIN
               rnumstid ← openlock(0, jflname($"rfcnumbers"));
               *tempsr* ← '0, *rfcnum*;
               IF (z1 ← namelook(rnumstid, $tempsr))= endfil THEN
               err($"RFC Number Error--Call NIC");
               rnumstid ← z1;
               rfctitle(FALSE, $lit, rnumstid);
               IF lit.L # 0 THEN
                  BEGIN
                  CCPOS *lit*;
                  IF FIND [':] CH ↑z1 ([/"INUSE"/] < [/NP/]>/ ) ↑z2  THEN
                     BEGIN
                     *lit* ← z1 z2;
                     setjtitle($lit);
                     END;
                  END;
               rfcdist(FALSE, $lit, rnumstid);
               IF lit.L # 0 THEN
                  BEGIN
                  CCPOS *lit*;
                  IF FIND [':] CH ↑z1 [EOL] < CH ↑z2 >  THEN
                     BEGIN
                     *lit* ← z1 z2;
                     setjdist($lit);
                     END;
                  END;
               close(rnumstid.stfile := 0);
               END;
         setjrfc($rfcnum); %set up RFC Number Field%
         END;
   RETURN(mesflg) END.

(dsubmit)PROC(number);
   LOCAL stype, mesflg;
   LOCAL STRING authstr[50];
   REF number;
   stype ← detjdocument(:mesflg); %get id of journal dicument..returns
   type and stid's in spec stack, and lit with literal if stype = llit%
   rstlit();
```

```
    DSP(←<  "Number:");
    IF lookc() = CA THEN
       BEGIN
       input(); %read past CA%
       DSP(↑OFF);
       getcnum($initsr, &number, $" JOURNAL ", 0, 0);
       dn(&number);
       *authstr* ← *initsr*; %ident of author%
       END
    ELSE
       IF NOT dckcnum(1, &number, $authstr, 0) THEN
          err($"Number In Use--unlock to use");
    %By here, number = number, authstr = author, stype = type, and
    b1,b2,lit contain parms depending on type%
    %Now open work file and set up header statement%
       setjwork(stype, &number, $authstr);
    RETURN(mesflg) END.

(getjdocument)PROC;
    %Get type and address of document for journal entry...
       Returrn type, and parms in b1, b2, lit depending on type%
    LOCAL stype, mesflg;
    mesflg ← 0;
    CASE inpcuc() OF
       ='?:
          BEGIN
          *lit* ← EOL,
             "File ", EOL,
             "Message ", EOL,
             "Statement ", EOL,
             "Plex ", EOL,
             "Branch ", EOL,
             "Group ", EOL,
             "Hard Copy";
          typeas($lit);
          GOTO STATE;
          END;
       =SP:
          BEGIN
          crlf();
          stype ← litv;
          BUMP mesflg
          END;
       ='B:
          BEGIN
          echo($"Branch ");
          stype ← brnchv
          END;
       ='F:
          BEGIN
          echo($"File ");
          stype ← filev
          END;
       ='G:
          BEGIN
          echo($"Group ");
```

```
        stype ← groupv
        END;
    ='H:
        BEGIN
        echo($"Hard Copy--Location: ");
        stype ← hcopyv;
        mesflg ← hcopyv;
        END;
    ='M:
        BEGIN
        echo($"Message ");
        crlf();
        stype ← litv ;
        BUMP mesflg
        END;
    ='P:
        BEGIN
        echo($"Plex ");
        stype ← plexv;
        END;
    ='S:
        BEGIN
        echo($"Statement ");
        stype ← stmtv;
        BUMP mesflg
        END;
ENDCASE error();
IF (stype = litv) OR (stype = hcopyv) THEN
    BEGIN %read literal%
    *xlit* ← NULL;
    txtlit($xlit);
    END
ELSE
    BEGIN
    tbug($jb1);
    IF stype = groupv THEN tbug($jb2)
    ELSE
        IF stype = filev THEN jb1.stpsid ← origin;
    END;
RETURN(stype, mesflg) END.

(detjdocument) PROC;
    %Get type and address of document for journal entry...
      Return type, and parms in b1, b2, lit depending on type%
    LOCAL stype, mesflg, litfg, oldfile;
    LOCAL STRING flnam[200];
    mesflg ← 0;
    DSP(  Submit ↑ Message);
    CASE inpcuc() OF
        =CA:
            BEGIN
            DSP( ↑OFF);
            stype ← litv;
            BUMP mesflg
            END;
        ='M,=SP: REPEAT(CA);
```

```
    ='B:
        BEGIN
        DSP( ... Branch);
        stype ← brnchv
        END;
    ='F:
        BEGIN
        DSP( ... File);
        stype ← filev
        END;
    ='G:
        BEGIN
        DSP( ... Group);
        stype ← groupv
        END;
    ='H:
        BEGIN
        DSP( ... "Hard Copy--Location:");
        stype ← hcopyv;
        mesflg ← hcopyv;
        END;
    ='P:
        BEGIN
        DSP( ... Plex);
        stype ← plexv;
        END;
    ='S:
        BEGIN
        DSP( ... Statement);
        stype ← stmtv;
        BUMP mesflg;
        END;
    ENDCASE
        BEGIN
        qm();
        GOTO STATE
        END;
IF (stype = litv) OR (stype = hcopyv) THEN %read literal%
    BEGIN
    *xlit* ← NULL;
    litfg ← FALSE;
    INPUT (SP BUG b1 BUG b2 CA litfg ← TRUE; / TEXT xlit CA);
    IF litfg THEN
        BEGIN
        tdr($b1,$b2,$b1,$b2);
        *xlit* ← b1 b2;
        END;
    rstlit();
    END
ELSE
    IF stype = filev THEN
        BEGIN
        DSP (...  File ↑ );
        oldfile ← [lda()].dacsp.stfile;
        *flnam* ← NULL;
        filnam (oldfile, $flnam);
```

```
        INPUT VISIBLE flnam CA;
        jbl.stfile ← clf ($flnam);
        jbl.stpsid ← origin;
        END
     ELSE
        BEGIN
        INPUT STID jbl;
        IF stype = groupv THEN INPUT STID jb2;
        END;
  RETURN (stype, mesflg);
  END.

(jdistd)PROC;
  %This procedure intractively handles the secondary distribution of
  JOurnal documents%
  %It assumes that the user is prepared to type in a Journal number at
  the time of the call%
  LOCAL STRING jnumber[5];
  IF tnumber($jnumber) < 0 OR curchr # CA THEN SIGNAL(1, $"Illegal
  Number");
  crlf();
  typeas($"TO: ");
  identlist($xlit);
  *xlit* ← *xlit*, ';; %terminator for getids%
  crlf();
  secdist($jnumber, $xlit);
  RETURN END.

(ddistd)PROC;
  %This procedure intractively handles the secondary distribution of
  JOurnal documents%
  %It assumes that the user is prepared to type in a Journal number at
  the time of the call%
  LOCAL char;
  LOCAL STRING jnumber[5];
  *jnumber* ← NULL;
  WHILE (char ← input()) IN ['0, '9] DO
     BEGIN
     *jnumber* ← *jnumber*, char;
     dn($jnumber);
     END;
  IF char # CA THEN SIGNAL(1,$"Illegal Number");
  DSP(< "TO:");
  didentlist($xlit, $"TO:");
  *xlit* ← *xlit*, '; ; %terminator for getids%
  secdist($jnumber, $xlit);
  RETURN END.

FINISH
```

JOEX

```
<NLS>JOEX.NLS;52, 15-OCT-71 11:46 WSD ;
FILE joex % 110 to <rel-nls>joex %
%Entry to Journal System%
   (setjwork)PROC(stype, number, authstr);
      %This procedure opens and initialises the Journal working file,
      and copies over the document so as to be te subplex of the
      initialised Journal Header Statement%
      LOCAL byt;
      LOCAL TEXT POINTER z1, z2;
      LOCAL STRING tempsr[50];
      REF number, authstr;
      %ton 12%
      !JSYS gjinf;
      gdname(r1, $tempsr);
      *tempsr* ← '<, *tempsr*, '>,  "JWORK", *initsr*;
      jworkstid ← openwk(0, $tempsr);
      %Set up default Sub-collection%
         ckident(&authstr, $auxlit, 0); %Get id record for this user%
         lgetsubcoll($auxlit, $tempsr, 0, 0);
      makeptr(jworkstid, $jwp1);
      ST jwp1← "(J", *number*,  %number%
         ") ; Author(s): /", *authstr*, %Author%
         "; Sub-Collections: ", *tempsr*, %Su-Collections%
         "; Clerk: ", *initsr*, %Clerk (operator)%
         ";  .IGD=0; .SNF=72; .MCH=65; .TABSTOPS=8,16,24,32,40,48,56,64;
          .PGN=-1; .SCR=2; .PES;", EOL, EOL; %Journal Formatting
         Directives%
      %Copy over the documnnt%
         CASE stype OF
            =litv: cis(jworkstid, $xlit, down);
            =stmtv: ccs(jworkstid, jb1, down);
            =brnchv: ccb(jworkstid, jb1, down);
            =groupv: ccg(jworkstid, jb1,  jb2, down);
            =plexv: ccp(jworkstid, jb1, down);
            =filev: BEGIN
               FIND jwp1 [EOL EOL] ↑z1 ←z1 ←z1;
               ST jwp1 ← SF(jwp1) z1, EOL, "Origin: ", SF(jb1) SE(jb1),
               z1 SE(z1);
               IF getsub(jb1) # jb1 THEN ccp(jworkstid, getsub(jb1),
               down);
               END;
            =hcopyv: BEGIN
               FIND jwp1 [')] ↑z1;
               ST z1 z1 ← " Hard Copy--Location: ", *xlit*, ';;
               END;
         ENDCASE err($"System Error (Illegal Type)");
      %toff 12%
      RETURN END.

%Parameter specification for Journal%
   (setjauthor)PROC(astr);
      LOCAL TEXT POINTER z1, z2;
      LOCAL STRING tempsr[100];
      LOCAL STRING oldaut[40];
      REF astr;
      astruc(&astr); %Set it upper case%
```

```
   CCPOS jwp1;
   FIND ["Author(s):"] [!/] ↑z1 [';] ↑z2 ←z2;
   *oldaut* ← z1 z2;
   ST jwp1 ← SF(jwp1) z1, *astr*, z2 SE(jwp1);
   IF *astr* # *oldaut* THEN
      BEGIN %Fix up subcollection stuff%
      FIND SF(*astr*) $NP ('↑/'&/) ↑z1 ([ '( / NP] ↑z2 ←z2 / $PT
      ↑z2);
      *astr* ← z1 z2;
      IF NOT ckident(&astr, $auxlit, 0) THEN
         BEGIN
         setjauthor($oldaut); %error--restore author field to initsr%
         err($"Illegal Author");
         END;
      lgetsubcoll($auxlit, $tempsr, 0, 0);
      setjsubcol($tempsr, jwp1);
      END;
   RETURN;
   END.

(setjdist)PROC(astr);
   REF astr;
   LOCAL TEXT POINTER z1,z2;
   FIND jwp1 (["Distribution: "] < CH [SP] ↑z1 > [';] ↑z2 /
      ["Author(s): "] [';] ↑z1 ↑z2);
   ST  z1 z2 ←  " Distribution: /", *astr*, ';;
   RETURN END.

(setjcomment)PROC(astr);
   REF astr;
   LOCAL TEXT POINTER z1,z2;
   FIND jwp1 ["Clerk: "] ["

   "] ↑z1;
   ST jwp1 ← SF(jwp1) z1, ".PEL;  .PGN=PGN-1; .GCR;", *astr*;
   RETURN END.

(setjclerk)PROC(astr);
   REF astr;
   LOCAL TEXT POINTER z1,z2;
   FIND jwp1 ["Clerk: "] ↑z1 [';] ↑z2 ←z2;
   ST jwp1 ← SF(jwp1) z1, *astr*, z2 SE(jwp1);
   RETURN END.

(setjexpedite)PROC(onoff);
   LOCAL TEXT POINTER z1,z2;
   FIND jwp1 ([" (Expedite) "] ↑z2 < CH [SP] > ↑z1/
      (["Title: "] < ['T] / ["Author(s): "] < ['A]) > ↑z1 ↑z2);
   IF onoff THEN ST z1 z2← " (Expedite) "
   ELSE ST z1 z2 ← NULL;
   RETURN END.

(setjkeyword)PROC(astr);
   REF astr;
   LOCAL TEXT POINTER z1,z2;
   FIND jwp1 (["Keywords:"] < [SP] ↑z1 > [';] ↑z2 /
```

```
      ([*Distribution: *]/[*Author(s): *]) [';] ↑z1 ↑z2);
   ST z1 z2 ← " Keywords: ", *astr*, ';;
   RETURN END.

(setjrfc)PROC(astr);
   REF astr;
   LOCAL TEXT POINTER z1,z2;
   FIND jwp1 ([*RFC# *] < CH [SP] ↑z1 > [';] ↑z2 /
      ([*Sub-Collections: *] / [*Keywords: *] / [*Distribution: *] /
      [*Author(s): *]) [';] ↑z1 ↑z2);
   ST z1 z2 ← " RFC# ", *astr*, ';;
   RETURN END.

(setjsubcol)PROC(astr,stid);
   LOCAL STRING tempsr[150];
   REF astr;
   LOCAL TEXT POINTER z1,z2;
   *tempsr* ← *astr*;
   astruc($tempsr); %Make sure that it is upper case%
   FIND SF(stid) ([*Sub-Collections: *] < CH [SP] ↑z1 > [';] ↑z2 /
      ( [*Keywords: *] / [*Distribution: *] / [*Author(s): *]) [';]
      ↑z1 ↑z2);
   ST z1 z2 ← " Sub-Collections: ", *tempsr*, ';;
   RETURN END.

(setjobsoletes)PROC(astr);
   LOCAL TEXT POINTER z1, z2;
   REF astr;
   FIND jwp1 ([*Obsoletes Document(s): *] < ['O] CH ↑z1 > [';] ↑z2/
      ([*RFC# *] / [*Sub-Collections: *] / [*Keywords: *] /
      [*Distribution: *] / [*Author(s): *]) [';] ↑z1 ↑z2);
   ST z1 z2 ← " Obsoletes Document(s): ", *astr*, '; ;
   RETURN;
   END.

(setjupdates)PROC(astr);
   LOCAL TEXT POINTER z1, z2;
   REF astr;
   FIND jwp1 ([*Updates Document(s): *] < ['U] CH ↑z1 > [';] ↑z2/
      ([*Obsoletes Document(s): *] / [*RFC# *] / [*Sub-Collections:
      *] / [*Keywords: *] / [*Distribution: *] / [*Author(s): *])
      [';] ↑z1 ↑z2);
   ST z1 z2 ← " Updates Document(s): ", *astr*, '; ;
   RETURN;
   END.

(setjtitle)PROC(astr);
   REF astr;
   LOCAL TEXT POINTER z1,z2;
   FIND jwp1 ([*  Title: .HED=*]  < [SP] [SP] ↑z1 > [';] ↑z2/
       [*Author(s):*]< [SP] ↑z1 ↑z2>);
   ST z1 z2 ← " Title: .HED=", '",  *astr*, '", ';;
   RETURN END.

%Status%
   (jstatus)PROC(stid, linkflg, linkptr, linkdir);
```

```
   LOCAL TEXT POINTER z1;
   REF linkptr, linkdir;
   %Type out status to user%
   makeptr(stid, $z1);
   *xlit*  ← "Catalog Number: ",
      *[getjnm($z1)]*,
      *[addeol(getjhcloc($z1))]*,
      *[addeol(getjauthor($z1))]*,
      *[addeol(getjtitle($z1))]*,
      *[addeol(getjdist($z1))]*,
      *[addeol(getjexp($z1))]*,
      *[addeol(getjkeyw($z1))]*,
      *[addeol(getjsubcol($z1))]*,
      *[addeol(getjrfc($z1))]*,
      *[addeol(getjobsoletes($z1))]*,
      *[addeol(getjupdates($z1))]*,
      *[addeol(getjclerk($z1))]*,
      *[addeol(plinkst(linkflg, &linkptr, &linkdir))]*,
      *[addeol(getjcomment($z1))]*;
   IF n1mode = fulldisplay THEN
      BEGIN
      *xlit* ← "


      ", *xlit*, "

      To Continue-- Type CA";
      supda(0);
      litdpy($xlit);
      END
   ELSE
      BEGIN
      dismes(2, $xlit);
      END;
   RETURN END.

(addeol)PROC(astr);
   %Accepts a string as an argument, an moves it into auxlit preceded
   by an EOL if it is not empty.  Returns address of auxlit or null
   string%
   REF astr;
   IF astr.L > 0 THEN
      BEGIN
      *lit2* ← EOL, *astr*;
      RETURN($lit2);
      END;
   RETURN($"");
   END.

(getjnm)PROC(ptr);
   REF ptr;
   IF NOT FIND ptr $NP '( ('J/'j) ↑p1 4$5 D ↑p2 THEN err($"Bad
   Journal Header");
   *auxlit* ← p1 p2;
   RETURN($auxlit);
   END.
```

```
(getjhcloc)PROCEDURE(ptr);
   REF ptr;
   IF FIND ptr ["Hard Copy--Location: "] ↑p1 [';] ↑p2 THEN
      BEGIN
      *auxlit* ← "Hard Copy--Location: ",p1 p2;
      RETURN($auxlit);
      END
   ELSE RETURN($"");
   END.

(getjauthor)PROCEDURE(ptr);
   REF ptr;
   FIND ptr ["Author(s):"] ['/] ↑p1 [';] ↑p2 ←p2;
   *auxlit* ← "Author(s): ",p1 p2;
   RETURN($auxlit);
   END.

(getjtitle)PROCEDURE(ptr);
   REF ptr;
   IF FIND ptr ["Title: "] ['"] -'" ↑p1 ←p1 ['"] ↑p2 ←p2 THEN
      BEGIN
      *auxlit* ← "Title: ",p1 p2;
      RETURN($auxlit);
      END
   ELSE RETURN($"");
   END.

(getjdist)PROCEDURE(ptr);
   REF ptr;
   IF FIND ptr ["Distribution: "] ['/] ↑p1 [';] ↑p2 ←p2 THEN
      BEGIN
      *auxlit* ← "Distribution: ",p1 p2;
      RETURN($auxlit);
      END
   ELSE RETURN($"");
   END.

(getjexp)PROCEDURE(ptr);
   REF ptr;
   IF FIND ptr ["(Expedite)"] THEN RETURN($"Expedited")
   ELSE RETURN($"");
   END.

(getjkeyw)PROCEDURE(ptr);
   REF ptr;
   IF FIND ptr ["Keywords: "] ↑p1 [';] ↑p2 ←p2 THEN
      BEGIN
      *auxlit* ← "Keywords: ",p1 p2;
      RETURN($auxlit);
      END
   ELSE RETURN($"");
   END.


(getjsubcol)PROCEDURE(ptr);
```

```
   LOCAL TEXT POINTER z1, z2, z3;
   LOCAL STRING tempsr[150], templsr[30];
   REF ptr;
   *auxlit* ← NULL;
   IF (FIND ptr ["Sub-Collections: "]   ↑p1 [';] ↑p2 ←p2 ) THEN
      *auxlit* ← p1 p2;
   IF rfcflg THEN *auxlit* ← "NWG ", *auxlit*;
   IF rfcflg AND NOT FIND BETWEEN p1 p2(["NIC"]) THEN
      *auxlit* ← "NIC ", *auxlit*;
   IF FIND ptr ["Distribution: "] ['/] ↑z1 THEN
      BEGIN
         *tempsr* ← NULL;
         getgpids($z1, $tempsr, 0); %Extract all group references
         from ident list%
         %Now fix up Sub-collection Field to reflect Distribution
         Groups%
            FIND SF(*tempsr*) ↑z1;
            WHILE (FIND z1 $NP ↑z2 1$PT ↑z1) DO
               BEGIN
               *templsr* ← +z2 z1;
               IF NOT FIND SF(*auxlit*) [*templsr*] THEN
                  *auxlit* ← *auxlit*, SP, *templsr*;
               END;
      END;
   IF auxlit.L > 0 THEN *auxlit* ← "Subcollections: ",*auxlit*;
   RETURN($auxlit);
   END.

(getjrfc)PROCEDURE(ptr);
   REF ptr;
   IF FIND ptr ["RFC# "]   ↑p1 [';] ↑p2 ←p2 THEN
      BEGIN
      *auxlit* ← "RFC Number = ",p1 p2;
      RETURN($auxlit);
      END
   ELSE RETURN($"");
   END.

(getjclerk)PROCEDURE(ptr);
   REF ptr;
   FIND ptr ["Clerk: "] ↑p1 ['.] ↑p2 ←p2;
   *auxlit* ← "Clerk: ",p1 p2;
   RETURN($auxlit);
   END.

(plinkst)PROC(linkflg, linkptr, linkdir);
   LOCAL STRING tempsr[30];
   REF linkptr, linkdir;
   IF NOT linkflg THEN RETURN($"");
   *tempsr* ← NULL;
   tshptr(linkptr, linkptr[1], $tempsr);
   *auxlit* ← "Place Link at ", *tempsr*, SP, *linkdir*;
   RETURN($auxlit);
   END.

(getjcomment)PROCEDURE(ptr);
```

```
      REF ptr;
      IF FIND ptr ["

      "] ["."GCR;"]  ↑pl THEN IF POS pl # SE(pl) THEN
         BEGIN
         *auxlit* ← "Comments: ",pl SE(pl);
         RETURN($auxlit);
         END;
      RETURN($"");
      END.

   (getjobsoletes)PROC(ptr);
      LOCAL TEXT POINTER z1, z2;
      REF ptr;
      IF FIND ptr ["Obsoletes Document(s): "] < ['0] > ↑z1 [';] ↑z2 ←z2
      THEN
         BEGIN
         *auxlit* ← z1 z2;
         RETURN($auxlit);
         END;
      RETURN($"");
      END.

   (getjupdates)PROC(ptr);
      LOCAL TEXT POINTER z1, z2;
      REF ptr;
      IF FIND ptr ["Updates Document(s): "] < ['U] >↑z1 [';] ↑z2 ←z2
      THEN
         BEGIN
         *auxlit* ← z1 z2;
         RETURN($auxlit);
         END;
      RETURN($"");
      END.

%Journal Entry Execution%
   (jouex)PROC(number,mesflg,linkflg, linkpl,linkdir);
      %This procedure does the actual journalising of the documents.
         It does not return, but rather calls reset to return control to
         TODAS/NLS when done%
      LOCAL usednum, rnumstid, jcstid, jnlstid, tstid;
      LOCAL STRING mfname[50], tempsr[10], modstr[100]; %message file
      name%
      REF number;
      %ton 13%
      %set stid's to null%
         jnlstid ← jcstid ← rnumstid ← orgstid;
      jolock(); %Check to make sure it will work%
      ON SIGNAL
         ELSE
            BEGIN
            sigclose(rnumstid.stfile := 0);
            sigclose(jcstid.stfile := 0);
            closeu(jnlstid.stfile := 0);
            jerror(sysmsg);
            END;
```

```
   IF NOT mesflg OR (mesflg = hcopyv)  THEN jouext(&number,mesflg,
   linkflg, linkpl,linkdir); %Until we get files straightened out%
   setjheader(&number, $modstr); %fill out journal header statement%
   IF NOT (jsavaccess ← access .A accmask[jrnlaccess]) THEN
      enablaccess(0, jrnlaccess);
   jcstid ← openlock(0, jflname($"tjcat"));
   jnlstid ← findactive(20, jcstid: usednum); %number should be
   number of statements in prospective journal document%
   entrjdoc(jnlstid, &number);
   closeu(jnlstid.stfile:=0);
   updjcat(&number, usednum, $mfname, jcstid); %update jcat file%
   *tempsr* ← 'J, *number*;
   jcatentry(IF jdebug THEN $"DUVALL" ELSE $"JOURNAL", $mfname,
   %file name%  $tempsr, FALSE, jcstid); %bbranch name%
   %Now insert Catalog Modification Commands as necessary%
      IF modstr.L > 0 THEN
         BEGIN
         IF (tstid ← namelook(jcstid, $"Catmods")) = endfil THEN
            BEGIN
            IF (tstid ← namelook(jcstid, $"jcatalog")) = endfil THEN
            badfil(jcstid.stfile);
            tstid ← cis(tstid, $"(catmods) Catalog Modifiction
            Commands", sucdir);
            END;
         cis(tstid, $modstr, down);
         END;
   close(jcstid.stfile := 0);
   distdoc($mfname, &number, TRUE); %distribute it%
   IF rfcflg THEN
      BEGIN %release RFC number%
      usedcnum($rfcnum, rnumstid ← openlock(0,
      jflname($"Rfcnumbers")));
      close(rnumstid.stfile := 0);
      END;
   usedcnum(&number, 0); %release used journal number%
   %now type out the link to the user%
      linkout(linkflg,linkpl,linkdir,$mfname);
   jreset();
   END.

(jouext)PROC(number,hcflag,linkflg,linkpl,linkdir);
   %This procedure does the actual journalising of the documents.
      It does not return, but rather calls reset to return control to
      TODAS/NLS when done%
   LOCAL usednum, rnumstid, jcstid, jdocstid, tstid;
   LOCAL STRING dfname[50], modstr[100]; %Document file name%
   REF number;
   %Set up SIGNAL statement%
      rnumstid ← jcstid ← jdocstid ← orgstid;
      ON SIGNAL ELSE
         BEGIN
         sigclose(rnumstid.stfile := 0);
         sigclose(jcstid.stfile := 0);
         close(jdocstid.stfile := 0);
         END;
   setjheader(&number, $modstr); %fill out journal header statement%
```

```
   IF NOT (jsavaccess ← access .A accmask[jrnlaccess]) THEN
      enablaccess(0, jrnlaccess);
   %Now open a work file with number as name%
      IF NOT hcflag THEN
         BEGIN
         jdocstid ← openwk(0, jflname(&number));
         %ton 5%
         setaccess(jdocstid.stfile, jrnlaccess);
         %Now fix up header in Journal file%
            ST jdocstid ← ",;
            ", SF(jworkstid) SE(jworkstid);
         ccp(jdocstid, getsub(jworkstid), down); %copy over plex of
         document%
         %toff 5%
         %ton 6%
         updtfl(jdocstid.stfile, 0); %update file%
         %toff 6%
         close(jdocstid.stfile := 0); %close and lock it%
         END;
   %Now update control file%
      jcstid ← openlock(0, jflname($"tjcat"));
      jcatentry(IF jdebug THEN $"DUVALL" ELSE $"JOURNAL", &number,
      $"0", hcflag, jcstid);
      %Now insert Catalog Modification Commands as necessary%
         IF modstr.L > 0 THEN
            BEGIN
            IF (tstid ← namelook(jcstid, $"Catmods")) = endfil THEN
               BEGIN
               IF (tstid ← namelook(jcstid, $"jcatalog")) = endfil
               THEN badfil(jcstid.stfile);
               tstid ← cis(tstid, $"(catmods) Catalog Modifiction
               Commands", sucdir);
               END;
            cis(tstid, $modstr, down);
            END;
      close(jcstid.stfile := 0);
   *dfname* ← *number*;
   distdoc($dfname, &number, hcflag); %distribute it%
   IF rfcflg THEN
      BEGIN
      usedcnum($rfcnum, rnumstid ← openlock(0,
      jflname($"Rfcnumbers")));
      close(rnumstid.stfile := 0);
      END;
   usedcnum(&number, 0); %release used journal number%
   %now type out the linnk to the user%
      linkout(linkflg,linkpl,linkdir, $dfname);
   jreset();
   END.

(linkout)PROC(linkflg,linkpl,linkdir, linkstr);
   %Insert link if so directed, and type it out to user%
   REF linkstr;
   IF nlmode = fulldisplay THEN
      *linkstr* ← *linkstr*, "--CA to continue";
   dismes(1,&linkstr);
```

```
    IF nlmode = fulldisplay THEN
        BEGIN
        UNTIL input() = CA DO NULL;
        dismes(0);
        recred();
        END;
    IF linkflg THEN xis(linkpl,&linkstr,linkdir);
    RETURN END.

(setlinkloc)PROCEDURE(onoff, linkpl,linkdir);
    REF linkpl,linkdir;
    IF onoff THEN
        IF nlmode = typewriter THEN
            BEGIN
            crlf();
            echon();
            typeas($"After location: ");
            tbug(&linkpl);
            levadj(linkpl,&linkdir);
            END
        ELSE
            BEGIN
            DSP(< "After location:" ↑);
            INPUT STID linkpl LEVADJ linkdir CA;
            END;
    RETURN(onoff) END.

%Catalog sub-collections%
(rdsubcoll)PROC(astr);
    %This procedure accepts an identlist from the input terminal, and
    subsequently checks that eahc each IDENT is that of a group.
        Assuming that all of this is legal, it returns the identlist in
        astrng%
    LOCAL idfile, idstid;
    LOCAL STRING tempsr[30], idinfo[500], modstr[100];
    REF astr;
    %Set up signal to close idfile%
        idfile ← 0;
        ON SIGNAL ELSE sigclose(idfile := 0);
    *astr* ← NULL;
    echon();
    idfile ← open(0, jflname($"identfile"));
    idstid ← 0;
    idstid.stpsid ← origin;
    idstid.stfile ← idfile;
    DO
        BEGIN
        *tempsr* ← NULL;
        rdident($tempsr, idstid);
        ckident($tempsr, $idinfo, idfile);
        IF NOT afgptst($idinfo, 0) THEN
            BEGIN
            crlf();
            typeas($"Individuals may not be used as Sub-Collection
            Names");
            crlf();
```

```
            IF astr.L = 0 THEN typeas($"Next id:") ELSE  typeas(&astr);
            REPEAT LOOP;
            END;
         *astr* ← *astr*, *tempsr*, SP;
         END UNTIL curchr = CA;
      IF astr.L > 0 THEN astr.L ← astr.L - 1;
      close(idfile := 0);
      RETURN;
      END.
FINISH
```

JOLIBE

```
<NLS>JOLIBE.NLS;69, 24-NOV-71 13:57 JDH ;
FILE jolibe  % LLO to <REL-nls>jolibe %
%Declarations%
   DECLARE jmaxsmt = 10000;
%Entry support%
   (setjheader)PROC(number, modstr);
      LOCAL TEXT POINTER idptr, z1, z2;
      LOCAL idfile, stid;
      LOCAL STRING adsubcols[100], oldsubcols[300], tempsr[30];
      REF number, modstr;
      %Number is Journal Number, modstr will be used for returning any
      catalog manipulation commands for other entries%
      %This routine fills out the journal header%
      %ton 3%
      idfile ← 0;
      ON SIGNAL ELSE sigclose(idfile := 0);
      %first fill in time and date fields%
         *datesr* ← NULL;
         !JSYS gtad;
         dtfrmt(r1, $datesr);
         CCPOS jwp1;
         IF NOT FIND ("; ;"/ ';] ↑p1) ←p1 ["Author(s):"] ['/] ↑p3 [';]
         ↑p4 ←p4 THEN err($"Journal Header Error");
         IF rfcflg THEN
            ST jwp1 ← SF(p1) p1, SP, *datesr*, "; .HJOURNAL=", '",
            "NWG/RFC# ", *rfcnum*, ".SPLIT;",  p3 p4, SP, *datesr*, "
            ", *number*, '",  p1 SE(p1)
         ELSE
            ST jwp1 ← SF(p1) p1, SP, *datesr*, "; .HJOURNAL=", '", p3
            p4, SP, *datesr*, "  ", *number*, '",  p1 SE(p1);
      %Now fill out name fields%
         %author field%
            *lit* ← NULL;
            %ton 0%
            idfile ←open(0,jflname($"identfile"));
            %toff 0%
            FIND jwp1 ["Author(s): "] ['/] ↑idptr;
            intids(0);
            LOOP
               BEGIN
               IF NOT getids($idptr, $lit, 1, idfile) THEN EXIT; %reads
               idents from jwp1 (up to ';) and puts info into $lit
               (appended)...returns FALSE if no idents left%
               *lit* ←*lit*,", ";
               END;
            IF lit.L > 0 THEN lit.L ← lit.L-2; %ignore last comma%
            CCPOS SF(jwp1);
            FIND ["Author(s): "] ↑jwp1;
            ST jwp1 ← SF(jwp1) jwp1, *lit*, jwp1 SE(jwp1);
         %Now do distributiion field%
            *lit* ← NULL;
            IF FIND jwp1 ["Distribution: "] ['/] ↑idptr THEN
               BEGIN
               intids(0);
               LOOP
                  BEGIN
```

```
                IF NOT getids($idptr, $lit, 1, idfile) THEN EXIT;
                %reads idents from jwpl (up to ';) and puts info into
                $lit (appended)...returns FALSE if no idents left%
                *lit* ←*lit*,", ";
                END;
             IF lit.L > 0 THEN lit.L ← lit.L-2;
             CCPOS SF(jwpl);
             FIND /"Distribution: "/ ↑jwpl;
             ST jwpl ← SF(jwpl) jwpl, *lit*, jwpl SE(jwpl);
             END;
          close(idfile := 0);
       %Now fix up Sub-collection Field to reflect RFC, Distribution
       Groups%
          IF FIND SF(jwpl) ↑jwpl SF(*/getjsubcol($jwpl)/*) /NP/ ↑z1 THEN
             BEGIN
             *oldsubcols* ← z1 SE(z1);
             setjsubcol($oldsubcols, jwpl);
             END;
       %Now build any catalog modification commands dictated by parameter
       fields in the header%
          IF &modstr THEN
             BEGIN
             %Obsoletes%
                *modstr* ← */getjobsoletes($jwpl)/*;
             %Updates%
                *modstr* ← *modstr*, EOL, */getjupdates($jwpl)/*;
             %Check to see if entries are NULL%
                IF modstr.L = 1 THEN *modstr* ← NULL
                ELSE *modstr* ← *number*, SP, *modstr*;
             END;
       %toff 3%
       RETURN END.

%Jcat manipulation%
   %J-catalog entries%
       (updjcat)PROC(number, numused, linkstr, jcstid);
          %This routine updates the jcat file, where numused is the total
          number of statements used in the active file.  It is expected
          that number contains tte journal number of the document just
          entered, and auxlit is used by  his routine%
          %jcstid is assumed to be the stid of the active file statement%
          LOCAL STRING tempsr/10/;
          LOCAL TEXT POINTER z1, z2;
          REF number, linkstr;
          %ton 7%
          IF (jcstid ← namelook(jcstid, $"ACTIVE")) = endfil THEN
          err($"Bad Jcat File");
          IF NOT FIND SF(jcstid) /"Used = "/ $NP ↑z1 1$D ↑z2 THEN
             err($"Bad Jcat File");
          *tempsr* ← STRING(numused); %Get around L10 Compiler glitch%
          ST z1 ← SF(z1) z1, *tempsr*, z2 SE(z2), " ", *number*;
          %Now save off name for distribution%
             IF NOT FIND SF(jcstid) /") ("/ ↑z1 /')/ ↑z2 ←z2 THEN
                err($"Bad Jcat File");
             *linkstr* ← z1 z2;
          %toff 7%
```

```
      RETURN;
      END.

   (jcatentry)PROC(astr1, astr2, astr3, hcflag, jcstid);
      LOCAL dir, stid, xstid;
      %make an entry in the catalog part of the jcat file.
         astr1 is user name, astr2 is file name for file, and astr3
         is branch name%
      LOCAL TEXT POINTER z1, z2;
      REF astr1, astr2, astr3;
      %ton 8%
      stid ← jcstid;
      IF (xstid ← namelook(stid, $"JCATALOG"))= endfil THEN err($"Bad
      Jcat File");
      IF (xstid := getsub(xstid)) = xstid THEN dir ← down
      ELSE
         BEGIN
         dir ← sucdir;
         xstid ← getail(xstid)
         END;
      xstid ← ccs(xstid, jworkstid, dir);
      stid ← xstid;
      xstid ← cis(xstid, &astr1, down);
      IF hcflag THEN
         BEGIN
         CCPOS SF(stid);
         FIND [/"Location: "/] ↑ z1 [/';/] ↑z2;
         ST xstid ← "Location of Document: ",  z1 z2;
         %toff 8%
         END
      ELSE
         ST xstid ← "Link to document: (", *astr1*, ', , *astr2*, ',
         , *astr3*, ":gw) ";
      %toff 8%
      RETURN END.

%Katalog Klean-up%
   (updtjo)PROC;
      %This procedure updates the pemanent copies of the Journal
      files JCAT an CNUMBERS from the temporary copies TCNUMBERS,
      TJCAT%
      LOCAL jcstid, tjcstid, cstid, tcstid, errflg;
      errflg ← jcstid ← tjcstid ← cstid ← tcstid ← 0;
      jsavaccess ← access;
      conjdir(TRUE);
      r1←getsbn($"JUPDCH");
      !JSYS setnm;
      STATE ← jcopst, spec;
      IF errflg THEN nlsrst()
      ELSE BUMP errflg;
      ON SIGNAL ELSE
         BEGIN
         undo(jcstid := 0);
         undo(tjcstid := 0);
         undo(cstid := 0);
         undo(tcstid := 0);
```

```
        access ← jsavaccess;
        conjdir(FALSE);
        rl←nlssbn;
        !JSYS setnm;
        END;
    enablaccess(0, jrnlaccess);
    %First do the JCAT file%
        jcstid ← openlock(0, jflname($"JCAT"));
        tjcstid ← openlock(0, jflname($"TJCAT"));
        updtpx(jcstid, tjcstid, $"Jcatalog", TRUE, FALSE);
        updtpx(jcstid, tjcstid, $"Jcatalog", TRUE, FALSE);
        updtpx(jcstid, tjcstid, $"catmods", TRUE, FALSE);
        closeu((jcstid:=0).stfile);
        closeu((tjcstid:=0).stfile);
    %Now do cnumbers%
        cstid ← openlock(0, jflname($"CNUMBERS"));
        tcstid ← openlock(0, jflname($"TCNUMBERS"));
        updtpx(cstid, tcstid, $"FREE", FALSE, FALSE);
        updtpx(cstid, tcstid, $"USED", FALSE, FALSE);
        updtpx(cstid, tcstid, $"INUSE", FALSE, FALSE);
        updtpx(cstid, tcstid, $"PREASSIGNED", FALSE, TRUE);
        closeu((cstid:=0).stfile);
        closeu((tcstid := 0).stfile);
    access ← jsavaccess; %restore access%
    rl←nlssbn;
    !JSYS setnm;
    nlsrst();
    END.

(updtpx)PROC(stid1, stid2, namest, tailf, replfg);
    LOCAL dir, tstid1, tstid2;
    REF namest;
    IF (tstid1 ← namelook(stid1, &namest)) = endfil THEN
    badfil(stid1.stfile);
    IF (tstid2 ← namelook(stid2, &namest)) = endfil THEN
    badfil(stid2.stfile);
    IF (stid2 ← getsub(tstid2)) # tstid2 THEN
        BEGIN %there is something to copy%
        IF NOT (tailf OR replfg) THEN
            BEGIN
            stid1 ← tstid1;
            dir ← down
            END
        ELSE
            IF ((stid1 ← getsub(tstid1)) = tstid1) THEN %nothing
               under bracnh ... move to stid1 in down dir% dir ← down
            ELSE
               BEGIN
               stid1 ← getail(stid1);
               dir ← sucdir
               END;
        IF replfg AND (stid1 # tstid1)  THEN crp(TRUE, stid1, stid2,
        0, 0)
        ELSE cmp(stid1, stid2, dir);
        END
    ELSE
```

```
            IF (stid1 ← getsub(tstid1)) = tstid1 THEN
               %replace% crs(TRUE, stid1, stid2, 0, 0);
         RETURN END.

      (undo)PROC(stid);
         %un-lock and close file%
         LOCAL fileno;
         IF (fileno ← stid.stfile) = 0 THEN RETURN;
         IF NOT writeout(fileno, $sar) OR NOT delpc(fileno, $sar)  THEN
            err($sar);
         close(fileno);
         RETURN END.

%Distribution stuff%
   (distdoc)PROC(astr, number, mesflg);
      %distribute document identified by jworkstid%
      LOCAL stid, idfile, dstid;
      LOCAL TEXT POINTER z1, z2, z3;
      LOCAL STRING linkuser[15];
      REF astr, number;
      %astr contains file name upon call, Return link to document in
      astr%
      %ton 9%
      dstid ← 0;
      ON SIGNAL ELSE
         BEGIN
         sigclose(dstid.stfile := 0);
         sigclose(idfile := 0);
         END;
      dstid ← ccs(getail(getsub(openlock(0, jflname($"distfile")))),
      jworkstid, sucdir);
      IF jdebug THEN *linkuser* ← "Duval1, " ELSE *linkuser* ← "Journal,
      ";
      CASE mesflg OF
         =0:  *astr* ← '(, *linkuser*, *astr*,  ", 1:w)";
         =1: *astr* ← '(, *linkuser*, *astr*, %journal file name%
            ", J", *number*, ":gw) ";
         =hcopyv:  *astr* ← "(Hard Copy--",  *linkuser*, *astr*,  ", )";
         ENDCASE err($"Illegal Message Type");
      IF mesflg # hcopyv THEN
         BEGIN
         FIND SF(dstid) ['')]  ↑z1;
         ST z1 z1 ← "  ", *astr*; %Insert link into header for
         distribution%
         END;
      %now insert individuals on distribution list%
         %ton 0%
         idfile ← open(0, jflname($"identfile"));
         %toff 0%
         IF FIND SF(dstid)["Distribution: "] ['//] ↑ z1 THEN
            distd1(dstid, $z1, idfile, IF mesflg = hcopyv THEN $"Hard
            Copy Document" ELSE 0);
      %now send author copies%
         IF FIND SF(dstid) ["Author(s):"] ['//] ↑ z1 THEN
            distd1(dstid, $z1, idfile, $"Author Copy")
         ELSE err($"System Error (No Author Field)");
```

```
      %toff 9%
   close(dstid.stfile := 0);
   close(idfile := 0);
   RETURN END.

(secdist)PROCEDURE(number, identlist);
   %Accepts a string containing a catalog number, and a string
   containing an identlist.
      Distributes the indicated Journal document to te persons on the
      list%
   LOCAL char, dstid, jcstid, dirflg;
   LOCAL TEXT POINTER z1, z2, z3;
   LOCAL STRING tempsr[10];
   REF number, identlist;
   jcstid  ← dstid ← orgstid;
   dirflg ← FALSE;
   ON SIGNAL ELSE
      BEGIN
      close(jcstid.stfile := 0);
      close(dstid.stfile := 0);
      IF dirflg THEN conjdir(FALSE); %Return him to directory%
      END;
   jcstid.stfile ← open(0, jflname($"jcat"));
   *tempsr* ← 'J, *number*;
   IF (z2  ← namelook(jcstid, $tempsr))= endfile THEN
      err( $"No Such Document");
   makeptr(asrref(&identlist), $z1);
   conjdir(TRUE); %Connect to Journal Directory%
   dirflg ← TRUE;
   dstid ← ccs(getail(getsub(dstid ← openlock(0,
   jflname($"distfile")))), z2, sucdir);
   z2 ← getsub(z2);
   IF FIND SF(z2) ["Link"] ['(] ↑z2 ←z2 THEN
      BEGIN
      FIND SF(dstid)[')] ↑z3;
      ST z3 z3 ← SP, z2 SE(z2); %link%
      END;
   close(jcstid.stfile := 0);
   FIND SE(dstid) > ↑z3;
   ST z3 z3 ← EOL, "Secondary Distribution Copy";
   CCPOS SF(dstid);
   IF FIND ["(Expedite)"] ↑z3 < ['(] ↑z2 >  THEN ST z2 z3 ← NULL;
   distd1(dstid, $z1, 0, $"(Secondary Distribution Copy)");
   close(dstid.stfile := 0);
   conjdir(FALSE);
   RETURN END.

(distd1)PROC(dstid, ptr, idfile, note);
   %This procedure does the actual manipulation of the distribution
   file to put entries in for the individual reciepients.
      dstid is te stid of the branch head in the distribution fule
      ptr is a pointer to the distribution list
      idfile is the identification file number (if non-zero)
      note is optionally a string which will be appended to the entry
      in the comments/notes field (in addition to any comments which
      may be in the distribution list).
```

```
        %
        LOCAL idfnum, stid;
        LOCAL TEXT POINTER z1, z2, z3;
        REF ptr, note;
        IF idfile = 0 THEN
           BEGIN
           idfnum ← 0;
           ON SIGNAL ELSE sigclose(idfnum := 0);
           idfnum ← open(0, jflname($"identfile"));
           END
        ELSE idfnum ← idfile;
        intids(0);
        LOOP
           BEGIN
           *lit* ← NULL;
           IF NOT getids(&ptr, $lit, 0, idfnum) THEN EXIT;
           getiid($lit, 0, $z1, $z2);
           *lit* ← '(, z1 z2, '); %Ident%
           IF FIND ptr < CH > ') THEN
              BEGIN
              IF
              NOT FIND < CH ↑z2  ['(] >  CH ↑z1 THEN
                 err($"System Error (Bad distribution List)");
              IF &note THEN
                 *lit* ← *lit*, EOL, "Note: ", z1 z2, EOL, *note*, ";;"
              ELSE
                 *lit* ← *lit*, EOL, "Note: ", z1 z2;
              END
           ELSE
              IF &note THEN *lit* ← *lit*, EOL, "Note: ", *note*, ";;";
           cis(dstid, $lit, down);
           END;
        IF NOT idfile THEN close(idfnum);
        RETURN END.

%Message files%
     (entrjdoc)PROC(stid, number);
        %Enter journal documment into active file as identified by stid%
        REF number;
        %ton 5%
        %first copy over document%
           stid ← getail(getsub(stid));
           ccb(stid, jworkstid, sucdir);
           stid.stpsid ← origin;
           ST SF(stid) ← SF(stid) SE(stid), " ", *number*; %update header%
        %toff 5%
        RETURN END.

     (findactive)PROC(numsmt, jcstid);
        %Find the actve journal file, given the required number of
        statements in numsmt, and jcstid is the stid of the origin
        statement of the journal control file.
        If there is no active journal file, or there is not enough room in
        the active journal file, then close it, and create a new active %
        LOCAL usedsmt, stid;
        LOCAL TEXT POINTER z1, z2;
```

```
   LOCAL STRING tempsr[15];
   %ton 4%
   %First look for active%
      IF (stid ← namelook(jcstid,$"ACTIVE"))= endfil THEN
      RETURN(newactive(jcstid), numsmt);%get new active file% %need
      new active file%
   %now check room here%
      %first get current number of statements in file%
         FIND SF(stid) ["Used = "] $NP ↑z1 $D ↑z2;
         *tempsr* ← z1 z2; %number of used statements%
         usedsmt ← VALUE($tempsr);
      %now check to see if room%
         IF usedsmt+numsmt > jmaxsmt THEN
            BEGIN %no more room in this active%
            relactive(stid, jcstid); %release it%
            RETURN(newactive(jcstid), numsmt);
            END
         ELSE RETURN(openactive(stid, FALSE), usedsmt+numsmt);
   END.

(newactive)PROC(jcstid);
   %This sets up a new active file, and opens it, returning the stid
   of the origin statement%
   LOCAL STRING tempsr[35];
   IF (jcstid := getail(getsub(jcstid))) = jcstid  THEN
      *tempsr* ←  "(ACTIVE) (JRNL1) Used = 0"
   ELSE
      BEGIN
      CCPOS SF(jcstid);
      IF NOT FIND $NP "(JRNL" ↑p1 1$D ↑p2 THEN
         SIGNAL(-5, $"Bad Entry in Jcat File"); %Make it look like a
         bad file%
      *tempsr* ← p1 p2;
      bumpstr($tempsr);
      *tempsr* ← "(ACTIVE) (JRNL", *tempsr*, ") Used = 0";
      END;
   jcstid ← cis(jcstid, $tempsr, sucdir);
   RETURN(openactive(jcstid, TRUE));
   END.

(openactive)PROC(stid, newfile);
   %This routine fetches the active name from the statement
   identified by stid, and opens the file, returning the stid of the
   origin statement%
   %Note that lock is not a concern because JCAT MUST BE OPEN which
   is locked%
   LOCAL STRING tempsr[15];
   CCPOS SF(stid);
   IF NOT FIND $NP "(ACTIVE) (" ↑p1 [')] ↑p2 ←p2 THEN
      err($"Jctl Error");
   *tempsr* ← p1 p2;
   IF newfile THEN
      BEGIN
      stid ← openwk(0, jflname($tempsr));
      setaccess(stid.stfile, jrnlaccess);
      closeu(stid.stfile);
```

```
        END;
      stid ← 0;
      stid ← openlock(0, jflname($tempsr));
      %toff 4%
      RETURN(stid);
      END.

   (relactive)PROC(stid);
      %This procedure releases the active identified in the jcat by
      stid%
      CCPOS SF(stid);
      FIND ["(ACTIVE) "] ↑p1;
      ST p1 ← p1 SE(p1);
      RETURN END.

%Error handling Routines%
   (jerror)PROC(errval);
      LOCAL count;
      jclosfl();
      IF errval < 0 THEN
         BEGIN
         r1 ← chbptr(empty) + $auxlit;
         r2 ← -errval .V 4B11;
         r3 ← 0;
         !JSYS erstr;
         GOTO baderr;
         GOTO baderr;
         count ← 0;
         (baderr): dismes(2,$"System Error");
         END
      ELSE dismes(2,errval);
      nlsrst();
      END.

%Routines for wrapping up JOurnal and returning to NLS%
   (jreset)PROC;
      jclosfl();
      %toff 13%
      IF jtimfg THEN jtime();
      nlsrst();
      END.

   (joctlc)PROC;
      %Come here on a control c%
      !JSYS 141B; %cis--clear intterupt system (can't us set because of
      duplicate symbol)%
      jclosfl();
      r1 ← -1; %exec%
      r2 ← 2B11; %↑C channel%
      !JSYS iic; %cause interrupt%
      r1 ← 2000; !JSYS disms; %for TENEX timing problem%
      dismes(2,$"Journal Process Aborted by ↑C");
      nlsrst();
      END.

%Routines dealing with files, directories, etc%
```

```
(jclosfl)PROC;
   %Close any journal files which may be open%
   IF jworkstid.stfile THEN close(jworkstid.stfile := 0);
   IF jcatstid.stfile THEN close(jcatstid.stfile := 0);
   IF jrnlstid.stfile THEN closeu(jrnlstid.stfile := 0);
   IF diststid.stfile THEN close(diststid.stfile := 0);
   IF numstid.stfile THEN close(numstid.stfile := 0);
   IF NOT jsavaccess THEN disabaccess(0, jrnlaccess);
   %connect back to original directory%
      conjdir(FALSE);
   %Set subsystem name back to regular nls%
      r1 ← nlssbn;
      !JSYS setnm;
   %De-activate ↑C channel%
      r1 ← 4B5;
      !JSYS rcm; %read channel mask--134%
      IF r1 .A 1B11 THEN
         BEGIN %de-activate it%
         r1 ← 4B5;
         r2 ← 1B11;
         !JSYS dic;
         r1 ← 3;
         !JSYS dti;
         END;
   RETURN END.

(jflname)PROC(astr);
   %This procedure accepts the name part of a Journal file name as a
   parameter, and constructs a full name from it.
   The extenxion is always .NLS, and the user under whom it is stored
   is Journal if it is running as the real system, and duvall
   otherwise (i.e. if jdebug is true)%
   %The file name is built up in the string 'jnamstr', whose address
   is returned%
   REF astr;
   IF jdebug THEN *jnamstr* ← "<DUVALL>"
   ELSE *jnamstr* ← "<JOURNAL>";
   *jnamstr* ← *jnamstr*, *astr*, ".NLS";
   RETURN($jnamstr);
   END.

(conjdir)PROC(cflag);
   %IF cflag true, connect to Journal Directory, saving info for
   curren one.   Flag false means connect back%
   LOCAL byt;
   IF cflag THEN
      BEGIN
      r3 ← byt ← chbptr(0)+$jpassw;
      !JSYS gtpsw; %read the password for connected directory%
      jpassw.L ← slngth(byt, r3);
      !JSYS gjinf;
      jdirn ← r2; %conncted directory number%
      r1 ← 1;
      r2 ← chbptr(0) + (IF jdebug THEN $"DUVALL" ELSE $"JOURNAL");
      !JSYS stdir;
      GOTO derr;
```

```
        GOTO derr;
        r1.lh ← 0;
        r2 ← chbptr(0) + (IF jdebug THEN $"WSD" ELSE $jnlpsw);
        IF NOT SKIP !JSYS cndir THEN
           (derr): err($"Directory Connect Failed");
        END
     ELSE
        BEGIN
         IF (r1 ← jdirn) = 0 THEN RETURN;
        r2 ← chbptr(0) + $jpassw;
        IF NOT SKIP !JSYS cndir THEN
           BEGIN
           !JSYS gjinf;
           r2 ← 0;
           IF NOT SKIP !JSYS cndir THEN
              err($"Connect return failed--left in Journal");
           err($"Connect return failed--returned to Login Directory");
           END;
        END;
     RETURN
     END.

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

  (lockjo)PROC(type);
     %Type = 1 for bad file lock (flag 1), and 0 for jlock (flag 0);
     %
     IF type > 1 THEN err($"Illegal Flag Number");
     flagut(type, $setfg);
     specttyout(0, $"Journal Locked");
     specttyout(30B, $"Journal Locked");
     RETURN;
     END.
  (unlkjo)PROC(type);
     %Type indates flag to be unlocked... -1 means all%
     IF type NOT IN [-1, 1] THEN err($"Illegal Flag Number");
     IF type # 1 THEN flagut(0, $rstfg);
     IF type # 0 THEN flagut(1,$rstfg);
     RETURN;
```

```
      END.
%Procedures for initialising Journal Entry Mode%
   (jntrint) PROCEDURE;
      %This procedure checks the journal flags and sets the system name
      upon entrance to the journal.%
      jdirn ← 0;
      IF jdebug THEN
         BEGIN
         crlf();
         typeas($"Experimental Journal System--Do not use");
         crlf();
         END
      ELSE
         IF jolock() THEN
            err($"Journal system temporarily not available");
      %set up subsystem name for statistics%
         IF NOT jnlsbn THEN jnlsbn ← <AUXCOD, getsbn>($jnlnam);
         r1 ← jnlsbn;
         !JSYS setnm;
      RETURN;
      END.

   (jctlcint) PROCEDURE;
      %This procedure performs the initialization required to service
      control C in the program.%
      r1 ← 4B5;
      !JSYS rpcap;
      r3 ← r2;
      !JSYS epcap; %Permits process to assign ↑C for pseudo interrupt%
      r1.lh ← 3;
      r1.rh ← 2;
      !JSYS ati;
      r1 ← 4B5;
      r2 ← 1B11;
      !JSYS aic; %Activate ↑C interrupt channel = 2%
      conjdir(TRUE);
      jt0 ← jt1 ← jt2 ← jt3 ← jt4 ← jt5 ← jt6 ← jt7 ← jt8 ← jt9 ← jt10 ←
      jt1 ← jt12 ← jt13 ← jt14 ← jt15 ← 0;
      RETURN;
      END.

%Timing Procedures%
   (jtime)PROC;
      %Type out jthe jtimes of jthe run, and wait for a command accept
      before proceeding%
      jtimout($"Open File", jt0);
      jtimout($"open lock", jt1);
      jtimout($"Open Work", jt2);
      jtimout($"Set J Header", jt3);
      jtimout($"Find Active", jt4);
      jtimout($"enter J doc", jt5);
      jtimout($"Update & delfn", jt6);
      jtimout($"Update JCAT", jt7);
      jtimout($"Jcat Entry", jt8);
      jtimout($"Dist Doc", jt9);
      jtimout($"check Ident", jt11);
```

```
    jtimout($"Set J Work", jt12);
    jtimout($"Get Number", jt10);
    jtimout($"Used Numbers", jt14);
    jtimout($"Check Number", jt15);
    jtimout($"Summed Total ",
    jt0+jt1+jt2+jt3+jt4+jt5+jt6+jt7+jt8+jt9+jt10+jt11+jt12+jt14+jt15);
    jtimout($"Elapsed Total", jt13);
    crlf();
    typeas($"(Type CA)");
    WHILE input() # CA DO NULL;
    RETURN END.

(jtimout)PROC(astr, value);
    REF astr;
    LOCAL STRING st[20];
    crlf();
    typeas(&astr);
    typeas($" = ");
    *st* ← STRING(value);
    typeas($st);
    RETURN END.

FINISH
%Programs for including--removing timing code%
  %This program will turn all coded timing statemnts into real ones
  which get compiled%
    PROGRAM p
      DECLARE STRING s1[10], s2[10];
      DECLARE TEXT POINTER z1, z2, z3, z4;
      (p)PROC;
        LOCAL offlg;
        IF NOT FIND ["ton"/"toff"] THEN RETURN(flag ← FALSE);
        FIND $NP  ↑z1 $LD ↑z2;
        offlg ← FIND SF(z1) ["toff"];
        *s1* ← 'j,  z1 z2;
        IF offlg THEN
          BEGIN
          ST z1 ← "%Time off % IF jtimfg THEN";
          z1 ← cis(z1, $"BEGIN", down);
          z1 ← cis(z1, $"!JSYS jobtm;", sucdir);
          z1 ← cis(z1, $" ", sucdir);
          ST z1 ← *s1*, " ← ", *s1*, "
          -jt0-jt1-jt2-jt3-jt4-jt5-jt6-jt7-jt8-jt9-jt10-jt11-jt12-jt
          13-jt14-jt15+r1;";
          cis(z1, $"END;", sucdir);
          END
        ELSE
          BEGIN
          ST z1 ← "%Time on % IF jtimfg THEN";
          z1 ← cis(z1, $"BEGIN", down);
          z1 ← cis(z1, $"!JSYS jobtm;", sucdir);
          z1 ← cis(z1, $" ", sucdir);
          ST z1 ← *s1*, " ← ", *s1*, "
          +jt0+jt1+jt2+jt3+jt4+jt5+jt6+jt7+jt8+jt9+jt10+jt11+jt12+jt
          13+jt14+jt15-r1;";
          cis(z1, $"END;", sucdir);
```

```
            END;
         %Now get rid of extra s1%
            FIND SF(z1) [*s1*] [*s1*] [*s1*] ↑z2 < [('+/'-)] > ↑z1;
            ST z1 z2 ← NULL;
         flag ← TRUE;
         RETURN;
         END.

      FINISH
%This program will turn real timing statments which get compiled back
into coded ones which do not%
   PROGRAM rp
      DECLARE STRING s1[10], s2[10];
      DECLARE TEXT POINTER z1, z2, z3, z4;
      (rp)PROC;
         IF NOT FIND ["%Time on %"/ "%Time off %"] ↑z1 THEN RETURN
         (flag ← FALSE);
         IF FIND SF(z1) ["off"] THEN *s1* ← "%toff " ELSE *s1* ←
         "%ton ";
         z2 ← getsuc(getsuc(getsub(z1)));
         CCPOS SF(z2);
         IF NOT FIND $NP "jt" ↑z2 1$D ↑z3 THEN RETURN(FALSE);
         ST z1 ← *s1*, z2 z3, '%;
         cdp(z2);
         RETURN(flag ← TRUE);
         END.

      FINISH
```

JUMP

```
<NLS>JUMP.NLS;96, 2-MAR-72 15:23 BLP ; /"gettgt"/;

FILE jump   %  L1O  to  <REL-NLS>JUMP  %

%......declarations......%

   % variable declarations %

      DECLARE EXTERNAL
         jshdrl = 4, %length jump stack header%
         jsentl = 4, %length jump stack entry%
         jsmxnt = 5, %maximum number of entries in jump stack%
         jsnbr = 5; %maximum number of jump stacks%
      DECLARE EXTERNAL
         lsentl = 3, %length link stack entry%
         lsmxnt = 5, %maximum number of link stack entries%
         lsasmx = 11; %max nbr words for string containg file
         name%
      DECLARE jagemax = 63, lagemax = 63;
      DECLARE gtjoif = 10402B1;
      DECLARE next = TRUE;
      DECLARE reset = FALSE, clear = TRUE;
      DECLARE new = 0, none = 0;
      DECLARE return = TRUE, ahead = FALSE;

%......command specification......%

   (newjmp)
      %This routine determines the jump command specified and
      transfers to the appropriate routine.%
      %--------------------%
      PROCEDURE;
      LOCAL char;
      zap();
      arm();
      LOOP
         BEGIN
         CASE (char ← lookc()) OF
            =SP, =CA, =D: NULL;
            ENDCASE input();
         CASE char OF
            ='a:  qja();
            ='b:  qjctrl(char);
            ='c:  qjstr(contnt);
            ='d:  qjctrl(char);
            ='e:  qje();
            ='f:  qjf();
            ='h:  qjctrl(char);
            ='i, =CA, =SP, =D:  qjctrl('i);
            ='j:  qjctrl('i);
            ='l:  qjl();
            ='n:  qjstr(seqname);
            ='o:  qjctrl(char);
            ='p:  qjctrl(char);
            ='r:  qjr();
            ='s:  qjctrl(char);
```

```
            ='t:  qjctrl(char);
            ='u:  qjctrl(char);
            ='w:  qjstr(word);
            =CD: GOTO STATE;
            ENDCASE qm();
         END;
      END.

%......intrafile jump control......%

   (qjctrl) % control for most jump commands %
      %This routine handles the jump commands that include
      structure specification in the command --- like down, up,
      etc.  It expects a charcter that specifies the command,
      finds the jump target, and recreates the display.  Control
      remains in the command until a new command is entered.%
      %--------------------%
      PROCEDURE(char);
      LOCAL tgtdpa, stid, srcdpa;
      REF tgtdpa;
      atg();
      LOOP
         BEGIN
         CASE char OF
            ='b: DSP(< Jump to ↑ Back);
            ='d: DSP(< Jump to ↑ Down);
            ='h: DSP(< Jump to ↑ Head);
            ='i: DSP(< Jump to ↑ Item);
            ='o: DSP(< Jump to ↑ Origin);
            ='p: DSP(< Jump to ↑ Predecessor);
            ='s: DSP(< Jump to ↑ Successor);
            ='t: DSP(< Jump to ↑ Tail);
            ='u: DSP(< Jump to ↑ Up);
            ENDCASE
               BEGIN
               char ← 'i;
               REPEAT;
               END;
         IF char # 'o THEN &tgtdpa ← spcitm( :stid)
         ELSE
            LOOP
               BEGIN
               stid ← origin;
               IF lookc() # CA THEN newjmp();
               srcdpa ← dsparea(lcda(input()));
               stid.stfile ← /srcdpa/.dacsp.stfile;
               disarm();
               IF &tgtdpa ← gettgt(srcdpa, stid) THEN EXIT LOOP;
               END;
         CASE char OF
            ='b: tgtdpa.dacsp ← getbck(tgtdpa.dacsp);
            ='d: tgtdpa.dacsp ← getsub(stid);
            ='h: tgtdpa.dacsp ← gethed(stid);
            ='i, ='o, = CA: tgtdpa.dacsp ← stid;
            ='p: tgtdpa.dacsp ← getprd(stid);
            ='s: tgtdpa.dacsp ←
```

```
            IF getftl(stid) THEN stid ELSE getsuc(stid);
         ='t: tgtdpa.dacsp ← getail(stid);
         ='u: tgtdpa.dacsp ← getup(stid);
         ENDCASE NULL;
      jmpdpy(&tgtdpa);
      IF char = 'o THEN char ← 'i;  %J origin becomes J item%
      END;
   END.

(qjstr)  % jump content/name/word %
   %This routine processes the jump name first and next, word first
   and next, and content first and next commands.  It expects an
   argument that indicates which flavor of jump it's processing: 1
   is name, 2 is word, and 3 is content.%
   %--------------------%
   PROCEDURE(type);
   LOCAL tgtdpa, stid, char;
   REF tgtdpa;
   atg();
   CASE type OF
      =seqname, =name: DSP(< Jump to Name ↑ First);
      =word: DSP(< Jump to Word ↑ First);
      =contnt: DSP(< Jump to Content ↑ First)
      ENDCASE NULL;
   CASE (char ← lookc()) OF
      ='n, ='f, =BC: input();
      ENDCASE;
   CASE char OF
      ='f, =SP, =BUG:
         BEGIN
         atg();
         CASE type OF
            =seqname, =name: DSP(< Jump to ↑ Name First);
            =word: DSP(< Jump to ↑ Word First);
            =contnt: DSP(< Jump to ↑ Content First)
            ENDCASE NULL;
         IF NOT &tgtdpa ← spcent(type, origin :stid) THEN
            REPEAT('f);
         tgtdpa.dacsp ← stid;
         jmpdpy(&tgtdpa);
         REPEAT('n);
         END;
      ='n:
         BEGIN
         atg();
         CASE type OF
            =seqname, =name: DSP(< Jump to ↑ Name Next);
            =word: DSP(< Jump to ↑ Word Next);
            =contnt: DSP(< Jump to ↑ Content Next)
            ENDCASE NULL;
         IF NOT &tgtdpa ← spcent(type, next :stid) THEN
            REPEAT('n);
         tgtdpa.dacsp ← stid;
         jmpdpy(&tgtdpa);
         REPEAT('n);
         END;
```

```
      =BC: qjctrl('i);
      ENDCASE newjmp ();
   END.

(qja) % jump ahead %
   %This is the JUMP AHEAD command.  The next entry is removed
   from the stack and that view is created by SPCDPY.  Control
   returns to the JUMP AHEAD command.%
   %--------------------%
   PROCEDURE;
   LOCAL dpa;
   LOOP
      BEGIN
      atg();
      DSP(... Ahead);
      IF jwait() THEN
         BEGIN
         dpa ← dsparea(lcda());
         bumpjs(dpa);
         jmpdpy(dpa);
         zap();
         END
      ELSE newjmp();
      END;
   END.

(qjr) % jump return %
   %This is the JUMP RETURN command.  The current view is
   pushed on the jump stack.  Then the previous view is popped
   off of the stack, and that view is created.  Control remains
   in the JUMP RETURN command.%
   %--------------------%
   PROCEDURE;
   LOCAL dpa;
   LOOP
      BEGIN
      atg();
      DSP(...Return);
      IF jwait() THEN
         BEGIN
         dpa ← dsparea(lcda());
         IF readjs(dpa) # [dpa].dacsp THEN pushjs(dpa);
         popjs(dpa);
         jmpdpy(dpa);
         END
      ELSE newjmp();
      END;
   END.

(qje) % jump end %
   %This procedure handles all of the Jump to End of ...
   commands.  If the item specified has an end that is not
   itself, control passes to Jump Item; otherwise, the display
   is recreated using the specified target, but control remains
   in the Jump End command.%
   %--------------------%
```

```
PROCEDURE;
LOCAL stid, dpa, char;
REF dpa;
atg();
DSP(<Jump to End of ↑ Item);
LOOP
   BEGIN
   CASE (char ← input()) OF
      ='c:
         BEGIN
         DSP(... Content);
         &dpa ← spcent(contnt, origin :stid);
         END;
      ='d:
         BEGIN
         DSP(... Down);
         &dpa ← spcitm( :stid);
         stid ← getsub(stid);
         END;
      ='h:
         BEGIN
         DSP(... Head);
         &dpa ← spcitm( :stid);
         stid ← gethed(stid);
         END;
      ='i:
         BEGIN
         DSP(... Item);
         &dpa ← spcitm( :stid);
         END;
      ='n:
         BEGIN
         DSP(... Name);
         &dpa ← spcent(name, origin :stid);
         END;
      ='o:
         BEGIN
         DSP(... Origin);
         &dpa ← spcitm( :stid);
         stid.stpsid ← origin;
         END;
      ='p:
         BEGIN
         DSP(... Predecessor);
         &dpa ← spcitm( :stid);
         stid ← getprd(stid);
         END;
      ='s:
         BEGIN
         DSP(... Successor);
         &dpa ← spcitm( :stid);
         stid ← IF getftl(stid) THEN stid ELSE getsuc(stid);
         IF stid.stpsid = origin THEN stid ← getsub(stid);
         END;
      ='t:
         BEGIN
```

```
            DSP(... Tail);
            &dpa ← spcitm( :stid);
            stid ← getail(stid);
            END;
         ='u:
            BEGIN
            DSP(... Up);
            &dpa ← spcitm( :stid);
            stid ← getup(stid);
            END;
         ='w:
            BEGIN
            DSP(... Word);
            &dpa ← spcent(word, origin :stid);
            END;
         =BC: newjmp();
         =CD: GOTO STATE;
         =CA: REPEAT ('i);
         ENDCASE
            BEGIN
            <NCTRL, caqm>();
            REPEAT;
            END;
      dpa.dacsp ← getvnd(stid, dpa.davspec.vslev);
      jmpdpy(&dpa);
      IF dpa.dacsp # stid THEN EXIT;
      END;
   qjctrl('i);
   END.

%......interfile jump control......%

   (qjl)
      %This statement is called by the JUMP LINK command.  When
      control returns to this routine, a new command has been
      specified.%
      %--------------------%
      PROCEDURE;
      DSP(...Link);
      qjlex();
      newjmp();
      END.

   (qjf)
      %This is the JUMP FILE command dispatch routine.
         JUMP FILE LINK operates essentially the same way as JUMP LINK.
         JUMP FILE RETURN pops one file element from the file jump
         stack and offers it in the name area.  A command accept
         results in jumping to that file, with the view set as it was.
         Any other character (except CD) results in the next element
         being popped from the file jump stack and offered in the name
         area.
         JUMP FILE AHEAD is similar to RETURN except that the file jump
         stack is read in the opposite direction.%
      %--------------------%
      PROCEDURE;
```

```
   LOCAL char,dpa;
   REF dpa;
   atg();
   LOOP
      BEGIN
      DSP(<Jump File ↑ Link);
      dn($"");
      CASE char ← input() OF
         =CA: REPEAT ('l)
         =CD: GOTO STATE
         =BC: newjmp();
         ='l:
            BEGIN
            DSP( ... Link);
            qjlex();
            END;
         ='r:
            BEGIN
            DSP( ... Return);
            CASE char ← input() OF
               =CA:
                  BEGIN
                  &dpa ← dsparea(lcda());
                  &dpa ← qjfra(&dpa, return);
                  jmpdpy(&dpa);
                  END;
               =CD: GOTO STATE;
               ENDCASE REPEAT 2(char);
            END;
         ='a:
            BEGIN
            DSP( ... Ahead);
            CASE char ← input() OF
               =CA:
                  BEGIN
                  &dpa ← dsparea(lcda());
                  &dpa ← qjfra(&dpa, ahead);
                  jmpdpy(&dpa);
                  END;
               =CD: GOTO STATE;
               ENDCASE REPEAT 2(char);
            END;
         ENDCASE qjctrl(char);
      END;
   END.

(qjlex)
   %This routine is used by the JUMP LINK and JUMP FILE LINK
   routines to get the link specification.  This routine calls
   LINKEX to perse the link and perform the necessary IO.  Control
   remains in this routine until the user enters something other
   than a bug or space as the first character of a link
   specification.%
   %--------------------%
   PROCEDURE;
   LOCAL TEXT POINTER ptr;
```

```
LOCAL olddpa, newdpa;
REF newdpa;
LOOP
   BEGIN
   CASE lookc() OF
      =CA: INPUT BUG ptr;
      =CD:
         BEGIN
         input();
         GOTO STATE;
         END;
      =SP:
         BEGIN
         af();
         input();
         *lit* ← NULL;
         IF lookc() # '( THEN *lit* ← '(;
         INPUT TEXT lit;
         IF *lit*[lit.L] # ') THEN *lit* ← *lit*, ');
         ptr.stfile ← 0;
         ptr.stastr ← TRUE;
         ptr.stadr ← $lit;
         ptr[1] ← 1;
         END;
      ENDCASE newjmp();
   olddpa ← dsparea(lcda());
   lnkspec(1, $ptr, $stno, $stn2, $stn, $num);
   IF stn2.L # empty THEN
      BEGIN
      dn($stn2);
      wait();
      END
   ELSE
      BEGIN
      *stn2* ← *[[[olddpa].dalink].lfilnm]*;
      IF stn.L # empty THEN
         BEGIN
         dn($stn);
         wait();
         END;
      END;
   &newdpa ← dsparea(lcda());
   IF NOT <IOCTL, compas> ($stn2, [newdpa.dalink].lfilnm)
      THEN linkex(&newdpa, $stn2, $stn, $num)
   ELSE
      BEGIN
      qjil(&newdpa, $stn, $num);
      IF <PRMSPC, invspc> (&newdpa) THEN
         <PRMSPC, putvspc> (&newdpa);
      END;
   *stn* ← NULL;
   dn($stn);
   jmpdpy(&newdpa);
   END;
END.
```

```
(lnkspec)
   %update comments%


   %This routine is used by both the TODAS and NLS link
   delimiter routines; it expects the most current position
   on the spec stack to contain a T-pointer that specifies
   where the search for a link should begin. It expects a
   parameter that tells which link after the T-pointer should
   be used.
      This routne calls QIFL to recreate the display if the
      link points to the file that the link appears in.
      Otherwise it pushes the current display start and vspecs
      onto the jump stack for the current file; opens the new
      file, pushes the file onto the link stack, and then
      pushes the new display start and vspecs onto the jump
      stack for the new file.
    It expects a second parameter that tells which display
   area the link appears in.%
   %--------------------%
   PROCEDURE(lnknum, tptadr, usrstg, fnmstg, stnstg, vspstg):
   REF  usrstg, fnmstg, stnstg, vspstg;
   <TXTEDT,lparse>(tptadr, $p1, $p2, $p3, $p4, $p5, $p6,
      $p7, $p8, lnknum);
   *usrstg* ← p1 p2; % user %
      astruc(&usrstg);
   *fnmstg* ← p3 p4; % file name %
      astruc(&fnmstg);
   *stnstg* ← p5 p6; % statement name, number, or marker %
      IF stnstg.L > empty AND *stnstg*[1] NOT= '# THEN
      astruc(&stnstg);
   *vspstg* ← p7 p8; % view %
   IF usrstg.L # empty AND fnmstg.L # empty THEN
      *fnmstg* ← '<, *usrstg*, '>, *fnmstg*
   ELSE IF NOT [tptadr].stastr AND fnmstg.L # empty THEN
      BEGIN
      <NLS, IOEXEC, gdftdir>( [tptadr].stfile, &usrstg);
      *fnmstg* ← '<, *usrstg*, '>, *fnmstg*;
      END;
   IF stnstg.L = empty THEN *stnstg* ← '0;
   RETURN;
   END.

(linkex)
   PROCEDURE(dpa, fnmstng, stnstg, vspstg);
   LOCAL stid, fno;
   stid ← nfstid(fnmstng, stnstg);
   uplkda(dpa, fnmstng, vspstg, stid);
   IF [flntadr(fno ← stid.stfile)].fllock THEN lockmes(fno);
      %Type out file locked message%
      %It is not clear that this is the proper place too type out
      this message, but it currenty seems to be the most central
      without being a core routine, and it already fiddles with
      display areas%
   RETURN;
   END.
```

```
(nfstid)
   PROCEDURE(fnmstng, stnstg);
   %Given two strings, the first containing a file name, the second
   a statement name, statement number, or marker, this routine will
   open the file, and return the corresponding stid.%
   %-------------%
   LOCAL fno;
   LOCAL TEXT POINTER tptr;
   LOCAL STRING lkmkst[15];
   REF dpa, stnstg;
   tptr ← origin;
   tptr.stfile ← fno ← jmpopn(fnmstng);
   IF stnstg.L > empty AND *stnstg*[1] = '# THEN %marker%
      BEGIN
      *lkmkst* ← *stnstg*[2 TO stnstg.L];
      tptr ← <INPFBK, lkmkr>($lkmkst, fno : tptr[1]);
      IF tptr = endfil THEN err($"No such marker");
      END
   ELSE %statement name or number%
      BEGIN
      specreg(&stnstg, name, $tptr);
      IF tptr = endfil THEN err($"No such name");
      END;
   RETURN(tptr);
   END.

(ofstid)
   %Given the display area address in order to get the file number
   and a string containing a statement name, statement number, or
   marker, this routine will open the file, and return the
   corresponding stid.%
   %--------------------%
   PROCEDURE(dpa, stnstg);
   LOCAL TEXT POINTER stptr;
   LOCAL STRING lkmkst[5];
   REF dpa, stnstg;
   stptr ← origin;
   stptr.stfile ← dpa.dacsp.stfile;
   IF stnstg.L > empty AND *stnstg*[1] = '# THEN %marker%
      BEGIN
      *lkmkst* ← *stnstg*[2 TO stnstg.L];
      stptr ← <INPFBK, lkmkr>($lkmkst, stptr.stfile : stptr[1]);
      IF stptr = endfil THEN err($"No such marker");
      END
   ELSE %statement name or number%
      BEGIN
      specreg(&stnstg, name, $stptr);
      IF stptr = endfil THEN err($"No such name");
      END;
   RETURN(stptr);
   END.

(uplkda)
   PROCEDURE(dpa, fnmstng, vspstg, stid);
   %Given the address of a display area, a string containing the
```

```
   name of the file, one containing a string of viewspecs, and an
   stid, this routine will update the display area's viewspecs,
   dacsp, and link stack to contain the current information.%
   %-------------%
   REF dpa;
   IF stid.stpsid = endfil THEN abort(1);
   pushjs(&dpa);
   pushls(&dpa, fnmstng);
   dpa.dacsp ← stid;
   dpa.daccnt ← empty;
   dpa.daempty ← FALSE;
   IF vspstg THEN feedlt(&dpa, vspstg);
   freflnt();
   RETURN;
   END.

(qjil)
   %This routine is used during the JUMP LINK command for an
   intra-file link in a single display area.  Given the address
   of a display area, this routine pushes the current view
   on the jump stack, and regenerates the display in accord
   with the link.
      This routine expects the name and num registers to be
      initialized as in LINKEX, (i.e., *stn* should contain
      the name of the destination, *num* an A-string containing
      the viewspecs).  If there is no name specified, then
      the current display start is used.%
   %--------------------%
   PROCEDURE(dpa, stnstg, vspstg);
   LOCAL  stid;
   LOCAL STRING lkmkst[5];
   REF dpa;
   stid ← ofstid(&dpa,stnstg);
   pushjs(&dpa);
   feedlt(&dpa, vspstg);
   dpa.dacsp ← stid;
   RETURN;
   END.

(qjfra)
   %This procedure is used during JUMP FILE RETURN and AHEAD.
   Given the address of a display window record, this routine
   will pop or bump the link stack until a command accept is
   encountered.  If the second argument is TRUE (= return),
   then the link stack is popped; otherwise it is bumped.
   It then senses the area in which the display is to be
   recreated, uses SETLS to update the target area's stacks,
   and then returns the address of the target display area.%
   %--------------------%
   PROCEDURE(srcdpa ,direction);
   LOCAL fileno, linkp, jumpp, tgtdpa;
   REF linkp, jumpp;
   IF NOT &linkp ← [srcdpa].dalink THEN abort(17);
   LOOP
      BEGIN
      &linkp ← IF direction = return THEN linkp.lsprev
```

```
        ELSE linkp.lsnext;
        dn(linkp.lfilnm);
        CASE input() OF
            =CA: EXIT;
            =CD: GOTO STATE;
            ENDCASE NULL;
        END;
    tgtdpa ← dsparea(lcda());
    pushjs(tgtdpa);
    fileno ← jmpopn(linkp.lfilnm);
    restjs(&linkp, fileno);
    IF srcdpa # tgtdpa THEN
        pushls(tgtdpa, linkp.lfilnm)
    ELSE [tgtdpa].dalink ← &linkp;
    setdpa(tgtdpa, [tgtdpa].dalink);
    freflnt();
    RETURN(tgtdpa);
    END.

%......target specification......%

    (spcent)
        %This procedure is used during jump name, word, and content
        processing.
            It offers the contents of CONREG in the statement name
            register, and permits the user to modify this string or choose
            a new one.
            It treats the string as a name if passed a 1 as the first
            argument, a a word, if passed a 2, and as content if passed a
            3.  SPCENT uses SPCREG to find the appropriate statement and
            put it on the spec stack.
            If the second argument is FALSE (= origin), the search for the
            target stid begins at the origin of the file indicated in the
            SRCDPA; otherwise it begins at SRCDPA.DACSP.
        It saves the string in CONREG and clears the name register. The
        user is given the chance to change viewspecs; the current view is
        pushed on the jump stack, and the new vspecs are saved in the
        target display area.  The address of the target display area and
        the target stid are returned.%
        %------------------%
        PROCEDURE(type, start);
        LOCAL TEXT POINTER stptr;
        LOCAL tgtdpa, srcdpa;
        REF srcdpa;
        LOOP
            BEGIN
            CASE type OF
                =seqname, =name:    *stn* ← *namreg*;
                =contnt, =word:    *stn* ← *conreg*;
                ENDCASE NULL;
            dn($stn);
            CASE lookc() OF
                ='a: %accept/edit proffered name%
                    BEGIN
                    input();
                    CASE type OF
```

```
            =seqname, =name: INPUT NAME stn CA;
            =word: INPUT WORD stn CA;
            =contnt: INPUT STRING stn CA;
            ENDCASE NULL;
         END
      =CA:  %bug text%
         BEGIN
         *stn*← NULL;
         CASE type OF
            =seqname, =name: INPUT NAME stn CA;
            =word: INPUT WORD stn CA;
            =contnt: INPUT STRING stn CA;
            ENDCASE NULL;
         END;
      =SP: %type in text%
         BEGIN
         input();
         af();
         *stn* ← NULL;
         dn($stn);
         CASE type OF
            =seqname, =name: INPUT NAME stn CA;
            =word: INPUT WORD stn CA;
            =contnt: INPUT STRING stn CA;
            ENDCASE NULL;
         END;
      =D: %statement number%
         BEGIN
         af();
         *stn* ← NULL;
         dn($stn);
         INPUT NAME stno CA;
         END;
      ENDCASE
         BEGIN
         *stn* ← NULL;
         dn($stn);
         newjmp();
         END;
   &srcdpa ← dsparea(lcda());
   stptr ← srcdpa.dacsp;
   stptr[1] ← 1;
   IF start = origin THEN stptr.stpsid ← origin
   ELSE IF type # seqname THEN stptr ← getnxt(stptr);
   specreg($stn, type, $stptr);
   IF stptr = endfil THEN
   CASE type OF
      =seqname, =name: abort(1);
      =word: abort(16);
      =contnt: abort(15);
      ENDCASE abort(0);
   IF tgtdpa ← gettgt(&srcdpa, stptr) THEN
      BEGIN
      CASE type OF
         =seqname, =name: *namreg* ← *stn*;
         =contnt, =word:
```

```
                BEGIN
                srchtype ← type;
                *conreg* ← *stn*;
                END;
             ENDCASE NULL;
          *stn* ← NULL;
          dn($stn);
          RETURN(tgtdpa, stptr);
          END
       ELSE bmoff();
       END;
   END.

(spcitm)
   %This routine is used for item recognition while processing jump
   commands.  A bug mark identifies the target statement; if the
   user enters a digit, the following string is interpreted as a
   statement number; if a space, a statement name.  Any other
   character is interpreted as a new jump command.
   The user is given the chance to change viewspecs; the current
   view is pushed on the jump stack, and the new vspecs are saved in
   the target display area.  The address of the target display area
   and the target stid are returned.%
   %--------------------%
   PROCEDURE;
   LOCAL srcdpa, tgtdpa;
   LOCAL TEXT POINTER stptr;
   REF srcdpa;
   LOOP
      BEGIN
      CASE lookc() OF
         =CA:
            BEGIN
            INPUT STID stptr;
            &srcdpa ← dsparea(lcda());
            END;
         =SP: %statement name%
            BEGIN
            *stn* ← NULL;
            input();
            INPUT NAME stn CA;
            &srcdpa ← dsparea(lcda());
            stptr ← srcdpa.dacsp;
            stptr.stpsid ← origin;
            specreg($stn, name, $stptr);
            IF stptr = endfil THEN abort(1);
            *stn* ← NULL;
            dn($stn);
            END;
         =D: %statement number%
            BEGIN
            *stno* ← NULL;
            INPUT NAME stno CA;
            &srcdpa ← dsparea(lcda());
            stptr ← srcdpa.dacsp;
            stptr.stpsid ← origin;
```

```
            specreg($stno, name, $stptr);
            IF stptr = endfil THEN abort(1);
            *stno* ← NULL;
            dn($stno);
            END;
         ENDCASE newjmp();
      disarm();
      IF tgtdpa ← gettgt(&srcdpa, stptr) THEN RETURN(tgtdpa, stptr)
      ELSE bmoff();
      END;
   END.

(gettgt)
   %Given a source display area and new stid, this routine will
   permit the user to change the viewspecs.  It then pushes the
   current view on the source jump stack, updates the target display
   area viewspecs.  If the user aborts the view specification, the
   routine returns FALSE; otherwise, it returns the address of the
   target display area.%
   %--------------------%
   PROCEDURE(srcdpa, stid);
   LOCAL tgtdpa;
   LOCAL STRING fname[100];
   REF srcdpa, tgtdpa;
   IF &tgtdpa ← invspc(&srcdpa) THEN
      BEGIN
      *fname* ← NULL;
      filnam(stid.stfile, $fname);
      IF *[[tgtdpa.dalink].lfilnm]* = *fname* THEN pushjs(&srcdpa)
      ELSE
         BEGIN
         pushjs(&srcdpa);
         pushls(&tgtdpa, $fname);
         END;
      putvspc(&tgtdpa);
      END;
   RETURN(&tgtdpa);
   END.

%.....find jump target.....%

(namelook)PROC(stid, astr);
   %This routine accepts an stid and astring, an finds the named
   statemet in the file indicated by the stid, retruning the stid
   oof the statemet or endfil.  Uses non-sequential lookup.  Copies
   name astring into local string so a literal may be used in the
   call%
   LOCAL TEXT POINTER z1;
   LOCAL STRING namest[40];
   REF astr;
   FIND SF(stid) ↑z1;
   *namest* ← *astr*;
   lookup($z1, $namest, name);
   RETURN(z1);
   END.
(lookup)PROCEDURE(ptr, astrng, type);
```

```
%THis routine accepts a pointer, string, and type, and does a
search through the file indicated by the pointer for the statemnt
indicated by the string and type as follows:
   type = name
      searches the file for the first statement found with the
      indicated name.
      Does a non-sequential search.
      The string is modified.
      Returns the stid of the statement as a result and in the
      pointer, or endfil on failure.
   type = nxtname
      Same as name, but starts from the place in the ring
      indicated by the stid in ptr.
      Note that this also is a non-sequential search
   type = seqname:
      Does a sequential search of the file for the next statemet
      (beginning with the one following the one indicated by ptr)
      with the indicated name.
      Returns stid in ptr, or endfil in ptr.
   type = contnt
      Does a sequential search of the file fo a statement with
      the content in string.
      Search starts with the character following the one
      indicated by ptr
      Returns same as seqname.
   type = contls
      Same as content, but looks only in the current statement
   type = word
      Searches for word in string in a manner equivalent to
      content.
      Returns same as content.
   type = sid
      &astrng is assumed to be an sid, and te file is searched
      for a statement with a matching sid.
      Returns same as name.
%
%-------------------%
LOCAL nhash, count, sidval, p1scn, p2scn, stid;
REF astrng, ptr;
IF ptr = endfil THEN RETURN(endfil);
CASE type OF
   = name, =nxtname:
      BEGIN
      astruc(&astrng);
      nhash ← hash(&astrng);
      stid ← IF type = nxtname THEN ptr ELSE 0;
      WHILE (stid ← lkupfast(ptr, stid, nhash, rnameh)) # endfil
      DO
         BEGIN
         CCPOS SF(stid);
         xtrnam($lkupreg, $swork, -1);
         IF *lkupreg* = *astrng* THEN
            RETURN(ptr ← stid);
         END;
      RETURN(ptr ← endfil);
      END;
```

```
      = seqname:
         BEGIN
         astruc(&astrng);
         nhash ← hash(&astrng);
         ptr[1] ← 1;
         ptr ← getnxt(ptr);
         END;
      = sid:
         RETURN(ptr ← lkupfast(ptr, 0, &astrng, rsid));
      = word, = contnt, = contls:
         % bump the pointer along one character %
         IF ptr[1] = empty THEN ptr[1] ← 1
         ELSE FIND ptr > CH ↑ptr;
      ENDCASE;
   UNTIL ptr = endfil DO
      BEGIN
      IF inptrf THEN %have a rubout%
         BEGIN
         ptr ← endfil;
         RETURN;
         END;
      CASE type OF
         =seqname:
            IF getnam(ptr) = nhash THEN
               BEGIN
               CCPOS SF(ptr);
               <UTILTY, xtrnam>($lkupreg, $swork, -1);
               IF <UTILTY, compas>($lkupreg, &astrng) THEN
               RETURN;
               END;
         =contnt:
            IF FIND ptr > [*astrng*] ↑ptr ←ptr THEN RETURN;
         =contls: %search is limited to a single statement%
            BEGIN
            IF NOT FIND ptr > [*astrng*] ↑ptr ←ptr THEN ptr ←
            endfil;
            RETURN;
            END;
         =word:
            BEGIN
            CCPOS ptr;
            LOOP
               IF FIND [*astrng*] ↑ptr ←ptr < THEN
                  BEGIN
                  count ← astrng.L;
                  UNTIL (count ← count-1) < empty DO FIND CH:
                  IF FIND ↑plscn -LD AND ptr > CH -LD THEN RETURN;
                  IF NOT FIND > plscn CH THEN EXIT;
                     % this can EXIT if astrng is empty and
                     have reached end of statement %
                  END
               ELSE EXIT;
            END;
         ENDCASE err($"NLS system error");
      ptr ← getnxt(ptr);
      ptr[1] ← 1;
```

```
      END;
   RETURN END.

(lkupfast)PROC(stid, start, value, field);
   LOCAL rnb, rnptr, rnt, pgindx, rngbkend, rngbkptr, fileno:
   REF rnptr, rngbkptr;
   fileno ← stid.stfile;
   rnt ← (rnb ← filehead[fileno]+$rngst-$filhed) + rngm;
   &rnptr ← rnb+start.stblk-1;
   IF start THEN
      BEGIN
      IF (start ← start.stwc + ring1) >= blksiz THEN  %Gone over to
      te next block%
         BEGIN
         BUMP &rnptr;
         start ← 0;
         END
      ELSE start ← start - fbhdl;
      END;
   WHILE (&rnptr ← &rnptr+1) < rnt DO
      IF rnptr.rfexis THEN
         BEGIN
         IF (pgindx ← rnptr.rfcore) = 0 THEN
            pgindx ← lodrfb(&rnptr-rnb+rngbas, rngtyp, fileno);
         rngbkend ← (&rngbkptr ← crpgad[pgindx] + fbhdl) + blksiz;
         IF start THEN &rngbkptr ← &rngbkptr + start := 0; %add  in
         displacement if it's there%
         WHILE &rngbkptr < rngbkend DO
            IF rngbkptr.field = value THEN
               BEGIN
               stid.stblk ← &rnptr-rnb;
               stid.stwc ← &rngbkptr-crpgad[pgindx];
               RETURN(stid);
               END
            ELSE &rngbkptr ← &rngbkptr + ring1;
         END;
   RETURN(endfil);
   END.

(getvnd) %find end of branch%
   % 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.

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

    (lsstat) PROCEDURE(astrng);  %puts file names in link stack into
    string passed%
        LOCAL  da, areano, lp, firstlp;
        REF astrng, da, lp;
        *astrng* ← NULL;
        areano ← 0;
        UNTIL (areano ← areano + 1) > dacnt DO
            IF (&da ← dsparea(areano)) THEN
                BEGIN
                IF astrng.L # empty THEN *astrng* ← *astrng*, EOL;
                IF (&lp ← firstlp ← da.dalink) THEN DO
                    *astrng* ← *astrng*, *[lp.lfilnm]*, EOL
                UNTIL (&lp ← lp.lsnext) = firstlp;
                END;
        RETURN;
        END.

    (jmpdpy)
        %This routine creates the display for jump and link
        routines.  It expects the address of a display area as an
        argument.%
        %------------------%
        PROCEDURE(dpa);
        LOCAL save;
        REF dpa;
        LOCAL STRING locstr[20];
        dpa.dadsp ← TRUE;
        dpa.daempty ← FALSE;
        save ← dpa.davspec.vsdaft := TRUE;
        <NLSEX, xdafrmt> (&dpa);
        dpa.davspec.vsdaft ← save;
        <INPFBK, dspvsp>(dpa.davspec, dpa.davspc2, 3);
        %UPDATE date-time area%
            <NLS, AUXCOD, getdat>($locstr);
            <NLS, INPFBK, ddt>($locstr);
        RETURN;
        END.

    (jmpopn) PROCEDURE(namstg);
        LOCAL jfn, jfnflg, fileno;
        REF namstg;
        jfnflg ← getgtjflg(read, origff, oldvrsn);
        IF NOT jfn ← lgetjfn (0, &namstg, $nlsext, jfnflg, $lit) THEN
            BEGIN
            *lit* ← "Cannot load file ", *namstg*, '(, *lit*, ');
            dismes(2, $lit);
            GOTO STATE;
            END;
        fileno ← open(jfn, &namstg);
        RETURN(fileno);
        END.
    (feedlt)  PROCEDURE(dpa, astrng);
```

```
    %Given the address of a display area, this routine changes
    its vspecs in accord with the specificaations in the
    A-string passed it.  It passes the characters in the
    A-string to <PRMSPC, SETLT>, except for content analyzer
    patterns.%
    %------------%
    LOCAL count, length, char, vs1, vs2;
    LOCAL TEXT POINTER tp;
    LOCAL STRING castng[40];
    REF dpa, astrng;
    length ← astrng.L;
    count ← empty - 1;
    vs1 ← dpa.davspec;
    vs2 ← dpa.davspc2;
    UNTIL (count ← count+1) > length DO
       IF (char ← *astrng*[count]) = '; THEN
          BEGIN
          *castng* ← NULL;
          UNTIL (char ← *astrng*[count ← count + 1]) = '; DO
             BEGIN
             *castng* ← *castng*, char;
             IF count >= length THEN EXIT LOOP1;
             END;
          *castng* ← *castng*, " ;;";
          FIND SF(*castng*) ↑tp;
          cgpc ($tp, &dpa);
          END
       ELSE vs1 ← <PRMSPC, setlt>(char, vs1, vs2 : vs2);
    dpa.davspec ← vs1;
    dpa.davspc2 ← vs2;
    RETURN;
    END.

(jwait) PROCEDURE;
    disarm();
    af();
    CASE lookc() OF
       =CD:
          BEGIN
          input();
          GOTO STATE;
          END;
       =CA: RETURN(input());
       ENDCASE RETURN(FALSE);
    END.


%.....intrafile (jump stack) utility routines.....%

    %These routines save and restore elements on the jump stack. The
    jump stack behaves like a ring, in that when the top of the stack is
    reached, it wraps around to the bottom.  The jump stacks are
    organized as follows:
       If the flag LJSMULT, in the link stack entry is TRUE, then
       the stack pointer, LJSHDR points to the the jump stack
       header for the file; if it is FALSE, then LJSHDR points
```

```
   to a 3 word area containing the most recent view of the
   file.
   The routine GETJS is used to get another jump stack; RELSJS
   frees a stack.
   When a jump stack is allocated to a file, it should be
   initialized so that the stid points to the origin and the
   viewspec words contain the standards assigned on entry to
   NLS (CLRJS does this).  When a link stack entry is reopened,
   the jump stack entry should be reset to include the new
   file number (RESTJS does this).%

(pushjs)
   %Given the address of a display area, this routine will
   push the csp and two viewspec words associated with the
   area onto the jump stack.
      If the link entry has no jump stack allocated to it,
      and the second argument is true, then this routine uses
      GETJS to allocate a jump stack.  Otherwise, if the link
      entry has no jump stack allocated to it, the stid and
      viewspecs are saved in the jump stack header area.
   If there is no link stack for this entry, the routine
   returns FALSE; otherwise it returns TRUE.%
   %--------------------%
   PROCEDURE(dpa);
   LOCAL linkp, jumpp;
   REF dpa, linkp, jumpp;
   IF NOT (&linkp ← dpa.dalink) THEN RETURN(FALSE);
   IF linkp.lsnew THEN
      BEGIN
      &jumpp ← linkp.ljshdr;
      linkp.lsnew ← FALSE;
      END
   ELSE
      BEGIN
      IF NOT linkp.ljsmult THEN getjs(&linkp);
      &jumpp ← pshjsp(&linkp);
      END;
   jumpp.jsstid ← dpa.dacsp;
   jumpp.jsvsp1 ← dpa.davspec;
   jumpp.jsvsp2 ← dpa.davspc2;
   jumpp.jsccnt ← dpa.daccnt;
   IF linkp.ljsmult AND [linkp.ljshdr].jsage
      THEN agejs(linkp.ljshdr);
   RETURN(TRUE);
   END.

(bumpjs)
   %Given the address of a display area, this routine will
   save the stid and two viewspec words from the next entry
   in the jump stack in the appropriate places in the display
   area.
      If the link entry has no jump stack allocated to it,
      the stid and viewspecs saved in the jump stack header
      area are saved.
   If there is no link stack for this entry, the routine
   returns FALSE; otherwise it returns TRUE.%
```

```
   %--------------------%
   PROCEDURE(dpa);
   LOCAL jumpp, linkp;
   REF jumpp, linkp, dpa;
   IF NOT &linkp ← dpa.dalink THEN RETURN(FALSE);
   DO &jumpp ← pshjsp(&linkp) UNTIL jumpp.jsstid # 0;
   setdpa(&dpa, dpa.dalink);
   IF linkp.ljsmult AND [linkp.ljshdr].jsage
      THEN agejs(linkp.ljshdr);
   RETURN(TRUE);
   END.

(popjs)
   %Given the address of a display area, this routine will
   save the stid and two viewspec words from the previous entry
   in the jump stack in the appropriate places in the display
   area.
      If the link entry has no jump stack allocated to it,
      the stid and viewspecs saved in the jump stack header
      area are saved.
   If there is no link stack for this entry, the routine
   returns FALSE; otherwise it returns TRUE.%
   %--------------------%
   PROCEDURE(dpa);
   LOCAL jumpp, linkp;
   REF jumpp, linkp, dpa;
   IF NOT &linkp ← dpa.dalink THEN RETURN(FALSE);
   DO &jumpp ← popjsp(&linkp) UNTIL jumpp.jsstid # 0;
   setdpa(&dpa, dpa.dalink);
   IF linkp.ljsmult AND [linkp.ljshdr].jsage
      THEN agejs(linkp.ljshdr);
   RETURN(TRUE);
   END.

(getret) PROCEDURE(dpa, count);
   %pops jump stack back number of times specified by count; returns
   stid on jump stack at that point, and character count; sets
   display area vspec words to those in new entry on jump stack;
   DOES NOT reset dacsp%
   LOCAL jumpp, linkp, stid, ccnt;
   REF jumpp, linkp, dpa;
   IF count <= 0 THEN
      BEGIN
      IF NOT &linkp ← dpa.dalink THEN abort(17);
      &jumpp ← IF NOT linkp.ljsmult THEN linkp.ljshdr
      ELSE [linkp.ljshdr].jscurr;
      IF NOT (stid ← jumpp.jsstid) THEN
         BEGIN
         stid ← dpa.dacsp;
         ccnt ← dpa.daccnt;
         END
      ELSE ccnt ← jumpp.jsccnt;
      RETURN(stid, ccnt);
      END;
   LOOP
      BEGIN
```

```
      &jumpp ← popjsp(dpa.dalink);
      IF (stid ← jumpp.jsstid) # 0 THEN
         IF (count ← count - 1) <= 0 THEN EXIT LOOP;
      END;
   dpa.davspec ← jumpp.jsvspl;
   dpa.davspc2 ← jsvsp2;
   ccnt ← jumpp.jsccnt;
   RETURN(stid, ccnt);
   END.

(getahd) PROCEDURE(dpa, count);
   %bumps jump stack ahead number of times specified by count;
   returns stid on jump stack at that point; sets display area vspec
   words to those in new entry on jump stack; DOES NOT reset dacsp%
   LOCAL jumpp, linkp, stid, ccnt;
   REF jumpp, linkp, dpa;
   IF count <= 0 THEN
      BEGIN
      IF NOT &linkp ← dpa.dalink THEN RETURN(FALSE);
      &jumpp ← IF NOT linkp.ljsmult THEN linkp.ljshdr
      ELSE [linkp.ljshdr].jscurr;
      IF NOT (stid ← jumpp.jsstid) THEN
         BEGIN
         stid ← dpa.dacsp;
         ccnt ← dpa.daccnt;
         END
      ELSE ccnt ← jumpp.jsccnt;
      RETURN(stid, ccnt);
      END;
   LOOP
      BEGIN
      &jumpp ← pshjsp(dpa.dalink);
      IF (stid ← jumpp.jsstid) # 0 THEN
         IF (count ← count - 1) <= 0 THEN EXIT LOOP;
      END;
   dpa.davspec ← jumpp.jsvspl;
   dpa.davspc2 ← jsvsp2;
   ccnt ← jumpp.jsccnt;
   RETURN(stid, ccnt);
   END.
(readjs)
   %Given the address of a display area, this routine will return
   the stid, two viewspec words, and current character count from
   the current entry in the jump stack.
      If the link entry has no jump stack allocated to it, the stid
      and viewspecs saved in the jump stack header area are saved.
   If there is no link stack for this entry, the routine returns
   FALSE; otherwise it returns TRUE.%
   %--------------------%
   PROCEDURE(dpa);
   LOCAL jumpp, linkp;
   REF jumpp, linkp, dpa;
   IF NOT &linkp ← dpa.dalink THEN RETURN(FALSE);
   &jumpp ← IF NOT linkp.ljsmult THEN linkp.ljshdr
   ELSE [linkp.ljshdr].jscurr;
   RETURN(jumpp.jsstid, jumpp.jsvspl, jumpp.jsvsp2, jumpp.jsccnt);
```

```
   END.

(writejs)
   %Given the address of a display area, an stid and two
   viewspec wordsthis routine will replace the currentjump
   stack entry with the values passed.
      If the link entry has no jump stack allocated to it,
      the stid and viewspecs saved in the jump stack header
      area are saved.
   If there is no link stack for this entry, the routine
   returns FALSE; otherwise it returns TRUE.%
   %--------------------%
   PROCEDURE(dpa, stid, vspc1, vspc2, ccnt);
   LOCAL jumpp, linkp;
   REF jumpp, linkp, dpa;
   IF NOT &linkp ← dpa.dalink THEN RETURN(FALSE);
   &jumpp ← IF NOT linkp.ljsmult THEN linkp.ljshdr
   ELSE /linkp.ljshdr/.jscurr;
   jumpp.jsstid ← stid;
   jumpp.jsvsp1 ← vspc1;
   jumpp.jsvsp2 ← vspc2;
   jumpp.jsccnt ← ccnt;
   RETURN(TRUE);
   END.

(popjsp)
   %Given the address of a link stack header, this routine
   will return the pointer to the previous element in the jump
   stackk, or the address of the single jump element (LJSHDR),
   if no stack is allocated to this file.  It saves the new
   pointer in JSCURR, if a stack is allocated to this file.%
   %--------------------%
   PROCEDURE(linkp);
   LOCAL jshead;
   REF jshead, linkp;
   IF NOT linkp.ljsmult THEN RETURN(linkp.ljshdr);
   &jshead ← linkp.ljshdr;
   jshead.jscurr ← jshead.jscurr - jsentl;
   IF jshead.jscurr < jshead.jsbotm THEN
      jshead.jscurr ← jshead.jsbotm + ((jsmxnt - 1) * jsentl);
   RETURN(jshead.jscurr);
   END.

(pshjsp)
   %Given the address of a link stack header, this routine
   will return the pointer to the next element in the jump
   stack, or the address of the single jump element (LJSHDR),
   if no stack is allocated to this file.  It saves the new
   pointer in JSCURR, if a stack is allocated to this file.%
   %--------------------%
   PROCEDURE(linkp);
   LOCAL jshead;
   REF jshead, linkp;
   IF NOT linkp.ljsmult THEN RETURN(linkp.ljshdr);
   &jshead ← linkp.ljshdr;
   jshead.jscurr ←jshead.jscurr + jsentl;
```

```
   IF jshead.jscurr >= jshead.jsbotm + (jsmxnt * jsentl) THEN
      jshead.jscurr ← jshead.jsbotm;
   RETURN(jshead.jscurr);
   END.

(restjs)
   %Given the address of a link stack entry and a file number,
   this routine resets the STID's in the jump stack to refer
   to the file number.%
   %--------------------%
   PROCEDURE(linkp, fileno);
   LOCAL jsfirst, jumpp;
   REF  jumpp, linkp;
   IF linkp.ljsmult THEN
      BEGIN
      &jumpp ← jsfirst ← [linkp.ljshdr].jscurr;
      DO jumpp.jsstid.stfile ← fileno
      UNTIL (&jumpp ← popjsp(&linkp)) = jsfirst;
      END
   ELSE [linkp.ljshdr].jsstid.stfile ← fileno;
   RETURN;
   END.

(clrjs)
   %Given the address of a link stack entry, this routine resets the
   STID's in the jump stack to refer to the origin of file 0.  The
   viewspecs are set to the standard viewspecs, and the character
   count to empty.%
   %--------------------%
   PROCEDURE (linkp);
   LOCAL jsfirst, jumpp;
   REF linkp, jumpp;
   IF linkp.ljsmult THEN
      BEGIN
      jsfirst ← &jumpp ← [linkp.ljshdr].jscurr;
      DO
         BEGIN
         jumpp.jsstid ← origin;
         jumpp.jsvspl ← stdvsp;
         jumpp.jsvsp2 ← stdvsp[1];
         jumpp.jsccnt ← empty;
         END
      UNTIL (&jumpp ← popjsp(&linkp)) = jsfirst;
      END
   ELSE
      BEGIN
      &jumpp ← linkp.ljshdr;
      jumpp.jsstid ← origin;
      jumpp.jsvspl ← stdvsp;
      jumpp.jsvsp2 ← stdvsp[1];
      jumpp.jsccnt ← empty;
      END;
   RETURN;
   END.

(getjs)
```

```
   %Given the address of a link stack entry, this routine will get a
   jump stack; it inserts the file view saved in LJSHDR as the first
   entry of the stack, and then creates the header information for
   the stack.
      If JSFREE is empty, then this routine finds the oldest jump
      stack, uses RELJS to release the stack.  Otherwise, it takes
      the stack indicated by JSFREE.%
   %--------------------%
   PROCEDURE(linkp);
   LOCAL jshdr, jumpp, oldjumpp, firstp;
   REF jshdr, linkp, jumpp, oldjumpp;
   WHILE NOT (&jumpp ← jsfree) DO
      BEGIN
      &jumpp ← &oldjumpp ← jsalloc;
      WHILE (&jumpp ← jumpp.jsnext) # 0 DO
         IF jumpp.jsage > oldjumpp.jsage THEN
            &oldjumpp ← &jumpp;
      freejs(oldjumpp.jslink);
      END;
   jsfree ← jumpp.jsnext;
   firstp ← &jumpp + ((jsmxnt * jsentl) - 1);
   DO (firstp) ← 0
   UNTIL (firstp ← firstp - 1) < &jumpp;
   &jshdr ← linkp.ljshdr;
   jumpp.jsstid ← jshdr.jsstid;
   jumpp.jsvsp1 ← jshdr.jsvsp1;
   jumpp.jsvsp2 ← jshdr.jsvsp2;
   jumpp.jsccnt ← jshdr.jsccnt;
   jshdr.jslink ← &linkp;
   jshdr.jsbotm ← jshdr.jscurr ← &jumpp;
   jshdr.jsnext ← jsalloc;
   jsalloc ← &jshdr;
   linkp.ljsmult ← TRUE;
   agejs(linkp.ljshdr);
   RETURN;
   END.

(freejs)
   %Given the address of a link stack entry, this routine will
   free its jump stack; it saves the current file view in
   LJSHDR.%
   %--------------------%
   PROCEDURE(linkp);
   LOCAL nxthdr, jshdr, loophd, jumpp, botstack;
   REF jshdr, loophd, linkp, jumpp;
   IF linkp.ljsmult THEN
      BEGIN
      &jshdr ← linkp.ljshdr;
      &jumpp ← jshdr.jscurr;
      botstack ← jshdr.jsbotm;
      IF (nxthdr ← jshdr.jsnext) # 0 THEN
         IF (&loophd ← jsalloc) # &jshdr THEN
            LOOP
               CASE loophd.jsnext OF
                  =0: EXIT LOOP;
                  =&jshdr:
```

```
                    BEGIN
                    loophd.jsnext ← nxthdr;
                    EXIT LOOP;
                    END;
                ENDCASE &loophd ← loophd.jsnext;
        jshdr.jsstid ← jumpp.jsstid;
        jshdr.jsvspl ← jumpp.jsvspl;
        jshdr.jsvsp2 ← jumpp.jsvsp2;
        jshdr.jsccnt ← jumpp.jsccnt;
        [botstack].jsnext ← jsfree;
        jsfree ← botstack;
        IF &jshdr = jsalloc THEN jsalloc ← nxthdr;
        linkp.ljsmult ← FALSE;
        END;
    RETURN;
    END.

(agejs)
    %Given the address of a jump stack header, this routine
    will make that jump stack the youngest, and age all others.
    If the stack is already the youngest, it will return without
    aging the others.%
    %--------------------%
    PROCEDURE(jshead);
    LOCAL jumpp;
    REF jshead, jumpp;
    IF jshead.jsage = new THEN RETURN;
    IF &jumpp ← jsalloc THEN
        DO IF jumpp.jsage < jagemax THEN BUMP jumpp.jsage
        UNTIL (&jumpp ← jumpp.jsnext) = none;
    jshead.jsage ← new;
    RETURN;
    END.

(setdpa)
    %Given the address of a display window and a link pointer.
    this routine will set the viewspec words and csp to the
    values indicated by the current jump stack for the link
    pointer.%
    %--------------------%
    PROCEDURE(dpa, linkp);
    LOCAL jumpp;
    REF jumpp, linkp, dpa;
    IF NOT (&linkp ← dpa.dalink) THEN abort(17);
    &jumpp ← IF linkp.ljsmult THEN [linkp.ljshdr].jscurr
        ELSE linkp.ljshdr;
    dpa.dacsp ← jumpp.jsstid;
    dpa.davspec ← jumpp.jsvspl;
    dpa.davspc2 ← jumpp.jsvsp2;
    dpa.daccnt ← jumpp.jsccnt;
    RETURN;
    END.

(setvsp)
    %Given the address of a display window and a link pointer.
    this routine will set the viewspec words to the
```

```
      values indicated by the current jump stack for the link
      pointer.%
      %--------------------%
      PROCEDURE(dpa, linkp);
      LOCAL jumpp;
      REF jumpp, linkp, dpa;
      IF NOT (&linkp ← dpa.dalink) THEN abort(17);
      &jumpp ← IF linkp.ljsmult THEN [linkp.ljshdr].jscurr
         ELSE linkp.ljshdr;
      dpa.davspec ← jumpp.jsvsp1;
      dpa.davspc2 ← jumpp.jsvsp2;
      dpa.daccnt ← jumpp.jsccnt;
      RETURN;
      END.

   (upjsccnt)
      %Given the address of a display window and a link pointer.
      this routine will set the viewspec words to the
      values indicated by the current jump stack for the link
      pointer.%
      %--------------------%
      PROCEDURE(dpa);
      LOCAL linkp, jumpp;
      REF jumpp, linkp, dpa;
      IF NOT (&linkp ← dpa.dalink) THEN abort(17);
      &jumpp ← IF linkp.ljsmult THEN [linkp.ljshdr].jscurr
         ELSE linkp.ljshdr;
      IF dpa.dacsp # jumpp.jsstid THEN pushjs(&dpa)
      ELSE jumpp.jsccnt ← dpa.daccnt;
      RETURN;
      END.

%......interfile (link stack) utility routines......%

   (bumpls)
      %Given the address of a display area, this routine will set
      DALINK to the next position (push) in the link stack.%
      %--------------------%
      PROCEDURE(dpa);
      LOCAL newlp;
      REF dpa;
      IF dpa.dalink THEN
         BEGIN
         agels(newlp ← [dpa.dalink].lsnext);
         RETURN(newlp);
         END
      ELSE RETURN(FALSE);
      END.

   (popls)
      %Given the address of a display area, this routine will
      set DALINK to the previous position (pop) in the link
      stack.%
      %--------------------%
      PROCEDURE(dpa);
      LOCAL newlp;
```

```
   REF dpa;
   IF dpa.dalink THEN
      BEGIN
      agels(newlp ← /dpa.dalink/.lsprev);
      RETURN(newlp);
      END
   ELSE RETURN(FALSE);
   END.

(pushls)
   %Given the address of a display area, and the address of
   an A-string that contains the file name, this routine will
   push that information onto the link stack.
      If this link stack contains the maximum number of
      entries, then the next entry in the ring is used for
      the link stack information.  Otherwise, GETLENT is used
      to get another entry.  This entry is inserted in the
      ring and filled with the current information.%
   %-------------------%
   PROCEDURE(dpa, astadr);
   LOCAL linkp, count;
   REF linkp, dpa, astadr;
   count ← 1;
   IF (&linkp ← dpa.dalink) THEN
      WHILE (&linkp ← linkp.lsnext) # dpa.dalink DO BUMP count;
   IF count < lsmxnt THEN &linkp ← getlent(dpa.dalink)
   ELSE
      BEGIN
      &linkp ← /dpa.dalink/.lsnext;
      freejs(&linkp);
      agels(&linkp);
      END;
   *//linkp.lfilnm//* ← *astadr*;
   linkp.lsnew ← TRUE;
   dpa.dalink ← &linkp;
   RETURN(dpa.dalink);
   END.

(readls)
   %Given the address of a display area and of a string, this
   routine will put the file name for the current link stack
   entry in the string (or null if there is none), and return%
   PROCEDURE(dpa, string);
   REF dpa, string;
   IF NOT dpa.dalink THEN *string * ← NULL
   ELSE *string* ← *//dpa.dalink/.lfilnm//*;
   RETURN;
   END.

(uplsfnm)
   PROCEDURE(ofnam, nfnam);  %change file name ofnam to nfnam in all
   link stacks%
   LOCAL areano, dpa, links;
   REF ofnam, nfnam, dpa;
   areano ← 0;
   UNTIL (areano ← areano + 1) > dacnt
```

```
   DO
      IF (&dpa ← dsparea(areano)) THEN
         IF (links ← dpa.dalink) THEN
            DO
               BEGIN
               IF *[[dpa.dalink].lfilnm]* = *ofnam* THEN
                  *[[dpa.dalink].lfilnm]* ← *nfnam*;
               END
            UNTIL (dpa.dalink ← bumpls(&dpa)) = links;
   RETURN;
   END.

(getlent)
   %Given the address of a link entry, this routine gets
   another link entry and inserts it in the ring after the
   link entry passed it.%
   %------------%
   PROCEDURE(linkp);
   LOCAL jshdr, dpa, areano, frstlp, templp, oldlp, oldda, age;
   REF jshdr, dpa, linkp, oldda, templp;
   IF &linkp THEN linkp.lsage ← 0;
   WHILE NOT (&templp ← lsfree) DO
      BEGIN
      oldlp ← age ← areano ← 0;
      UNTIL (areano ← areano + 1) > dacnt DO
      IF (&dpa ← dsparea(areano)) THEN
         BEGIN
         IF (frstlp ← &templp ← dpa.dalink) THEN
         DO
            IF templp.lsage > age THEN
               BEGIN
               &oldda ← dpa;
               oldlp ← &templp;
               age ← templp.lsage;
               END
         UNTIL (&templp ← templp.lsnext) = frstlp;
         END;
      IF oldda.dalink = oldlp THEN oldda.dalink ←
      freelent(oldlp)
      ELSE freelent(oldlp);
      END;
   lsfree ← templp.lsnext;
   IF &linkp THEN
      BEGIN
      [templp.lsnext ← linkp.lsnext].lsprev ← &templp;
      linkp.lsnext ← &templp;
      templp.lsprev ← &linkp;
      END
   ELSE templp.lsnext ← templp.lsprev ← &templp;
   &jshdr ← templp.ljshdr;
   jshdr.jsstid.stpsid ← origin;
   jshdr.jsvsp1 ← stdvsp;
   jshdr.jsvsp2 ← stdvsp[1];
   templp.ljsmult ← FALSE;
   templp.lsnew ← TRUE;
   agels(&templp);
```

```
   RETURN(&templp);
   END.

(freelent)
   %Given the address of a link stack entry, this routine will
   return it to the link stack free list, and update the
   appropriate pointers in the stack.
      If this is the only entry in the ring, the routine
      returns FALSE; otherwise, it returns the address of the
      next link entry in the ring.%
   %------------%
   PROCEDURE(linkp);
   LOCAL temp;
   REF linkp;
   IF linkp.ljsmult THEN freejs(&linkp);
   IF linkp.lsnext = &linkp THEN
      BEGIN
      linkp.lsnext ← lsfree;
      lsfree ← &linkp;
      RETURN(FALSE);
      END;
   temp ← [linkp.lsprev].lsnext ← linkp.lsnext;
   [linkp.lsnext].lsprev ← linkp.lsprev;
   linkp.lsnext ← lsfree;
   lsfree ← &linkp;
   RETURN(temp);
   END.

(agels)
   %Given the address of a link stack entry, this routine will
   make that entry the youngest and age all other entries.
   If the entry is already the youngesst, this routine will
   not age the others.%
   %------------%
   PROCEDURE(linkp);
   LOCAL areano, templinkp, firstlinkp, dpa;
   REF linkp, templinkp, firstlinkp, dpa;
   areano ← 0;
   UNTIL (areano ← areano + 1) > dacnt DO
   IF (&dpa ← dsparea(areano)) AND dpa.daexis THEN
      BEGIN
      IF (&firstlinkp ← &templinkp ← dpa.dalink) THEN
         DO
            IF templinkp.lsage < lagemax THEN BUMP templinkp.lsage
         UNTIL (&templinkp ← templinkp.lsnext) = &firstlinkp;
      END;
   linkp.lsage ← new;
   RETURN;
   END.

FINISH of JUMP
```

LITDSP

```
<NLS>LITDSP.NLS;2, 30-DEC-71 17:02 MSC ;
FILE litdsp
   %.......Global REF's.....%
      REF litda;
   %.........Declarations.....%
      DECLARE STRING
         ltdem1 = "Cannot display more text.  Enter CA to save what you
         have.",
         ltdem2 = "Display area tables inconsistent.  Unable to accept
         text.";
      DECLARE suppress = 1, restore = 0;
      REGISTER r1 = 1;
   (litdpy) PROCEDURE(astrng);  %display string in text area%
      %--------------%
      LOCAL
         newls, tmp, astbpe, astbps, ls,
         count, lclbp, lsstrt, col, char, newline;
      REF  newls, astrng, ls;
      &ls ← IF litda.dacsp = endfil THEN litdnt(&astrng)
         ELSE aplcls();
      CASE astrng.L OF
         <litda.daccnt:
            BEGIN
            astbpe ← chbptr(astrng.L) + &astrng;
            astbps ← chbptr(empty) + &astrng;
            WHILE (lsstrt ← slngth(astbps, ls.rtbps)) > astrng.L DO
               IF NOT (&ls ← aplrls(&ls)) THEN
                  BEGIN
                  &ls ← getrta(litda.darts);
                  EXIT LOOP 1;
                  END;
            IF *astrng*[astrng.L] = NP THEN
               ls.rtstid ← astbpe
            ELSE
               BEGIN  %find line break character, this line segment%
               ls.rtstid ← ls.rtbps;
               UNTIL (lsstrt := lsstrt + 1) >= astrng.L DO
                  IF *astrng*[lsstrt] = NP THEN
                     ls.rtstid ← chbptr(lsstrt) + &astrng;
               END;
            ls.rtbpe ← astbpe;
            litda.daccol ← ls.rtlcol ←
               ls.rtfcol + slngth(ls.rtbps, astbpe);
            litda.daccnt ← astrng.L;
            wrtstr(&litda, &ls);
            END;
         > litda.daccnt:
            BEGIN
            count ← litda.daccnt + 1;
            LOOP
               BEGIN
               CASE (char ← *astrng*[count]) OF
                  =CR, =EOL:
                     BEGIN
                     lclbp ← ls.rtstid ← aplchr(&ls);
                     wrtstr(&litda, &ls);
```

```
            IF NOT (&ls ← aplgls(&ls, TRUE)) THEN RETURN;
            ls.rtstid ← ls.rtbpe ← ls.rtbps ← lclbp;
            litda.daccol ← ls.rtfcol ← ls.rtlcol ← 1;
            END;
         =TAB:
            BEGIN
            col ←
               fndtab(&litda, (litda.daccol-1)*litda.dahinc) /
                  litda.dahinc;
            lclbp ← ls.rtstid ← aplchr(&ls);
            IF col > litda.damcol THEN
               BEGIN
               ls.rtlcol ← litda.damcol;
               litda.daccol ← 1;
               BUMP litda.dacrow;
               newline ← TRUE;
               END
            ELSE
               BEGIN
               ls.rtlcol ← col;
               newline ← FALSE;
               litda.daccol ← col;
               END;
            wrtstr(&litda, &ls);
            &ls ← aplgls(&ls, newline);
            ls.rtstid ← ls.rtbpe ← ls.rtbps ← lclbp;
            END;
         =SP: ls.rtstid ← aplchr(&ls);
         =PT: aplchr(&ls);
         ENDCASE NULL;
      IF litda.daccol >= litda.damcol THEN
         BEGIN  %line break not at beginning of line%
         IF ls.rtstid # ls.rtbps THEN
            BEGIN
            IF NOT (&newls ← aplgls(&ls, TRUE)) THEN RETURN;
            ls.rtlcol ← ls.rtfcol +
               slngth(ls.rtbps, ls.rtstid);
            newls.rtbpe ← ls.rtbpe := newls.rtstid ←
               newls.rtbps ← ls.rtstid;
            newls.rtfcol ← 1;
            newls.rtlcol ← newls.rtfcol +
               slngth(newls.rtbps, newls.rtbpe);
            litda.daccol ← 1;
            wrtstr(&litda, &ls);
            wrtstr(&litda, &newls);
            &ls ← &newls;
            END
         ELSE
            BEGIN
            IF ls.rtfcol # 1 THEN %move this segment to next
            line%
               BEGIN
               ls.rtlcol ← ls.rtlcol - ls.rtfcol;
               ls.rtfcol ← 1;
               ls.rtx1 ← litda.daleft;
               ls.rtx2 ← litda.daright;
```

```
                    ls.rty ← ls.rty + litda.davinc;
                    &newls ← &ls +lsrtl;
                    newls.rty ← newls.rty + litda.davinc;
                    litda.daccol ← ls.rtlcol;
                    END
                 ELSE
                    BEGIN
                    IF NOT (&ls ← aplgls(&ls, TRUE)) THEN RETURN;
                    litda.daccol ← l;
                    END;
                 wrtstr(&litda, &ls);
                 END;
              END;
           IF (count ← count + 1) > astrng.L THEN EXIT LOOP 1;
           END;
        END;
     ENDCASE NULL;
  litda.daccnt ← astrng.L;
  wrtstr(&litda, &ls);
  RETURN;
  END.


(litdnt) PROCEDURE(astrng);
  %initialize litda to display astrng in literal display area.%
  LOCAL firstls, blnkls;
  REF firstls, blnkls;
  litda.dacsp ← astrng;
  litda.dacsp.stastr ← TRUE;
  litda.daccnt ← empty;
  litda.daccol ← 1;
  litda.dacrow ← 1;
  litda.damcol ← (litda.daright - litda.daleft)/litda.dahinc;
  &firstls ← getrta(litda.darts);
  &blnkls ← &firstls + lsrtl;
  firstls.rtnew ← firstls.rtexis ← TRUE;
  firstls.rtllsg ← FALSE;
  blnkls.rtnew ← FALSE;
  blnkls.rtllsg ← blnkls.rtexis ← TRUE;
  firstls.rtfcol ← firstls.rtlcol ←
     blnkls.rtfcol ← blnkls.rtlcol ← 1;
  firstls.rtbps ← firstls.rtbpe ← firstls.rtstid ←
     blnkls.rtbps ← blnkls.rtbpe ← blnkls.rtstid ←
     chbptr(empty) + astrng;
  firstls.rtx1 ← blnkls.rtx1 ← litda.daleft;
  firstls.rtx2 ← blnkls.rtx2 ← litda.daright;
  firstls.rty ← 1;
  blnkls.rty ← firstls.rty + litda.davinc;
  blnkls.rtfont ← firstls.rtfont ← litda.dafont;
  blnkls.rtcsize ← firstls.rtcsize ← litda.dacsize;
  blnkls.rthinc ← firstls.rthinc ← litda.dahinc;
  aplsrd(&firstls, suppress);
  aplsrd(&blnkls, suppress);
  RETURN(&firstls);
  END.
```

```
(aplchr) PROCEDURE(ls);
   %Given a line segment rt address, this rountine wil bump the
   requisite pointers to point to the next character in the buffer.
   It return the updated byte pointer.%
   %------------%
   LOCAL  temp, lclbp;
   REF ls;
   lclbp ← ls.rtbpe;
   temp ← ↑lclbp;
   ls.rtbpe ← lclbp;
   BUMP ls.rtlcol, litda.daccol;
   RETURN(lclbp);
   END.


(aplcls) PROCEDURE;
   %returns address of current line segment for literal display area%
   LOCAL ls, count;
   REF ls;
   count ← litda.darts;
   DO
      BEGIN
      &ls ← getrta(count);
      IF NOT ls.rtexis OR ls.rtllsg THEN
         BEGIN
         dismes(2, $"Literal display area tables destroyed; unable to
         accept text");
         GOTO STATE;
         END;
      IF ls.rtnew THEN RETURN(&ls);
      END
   WHILE (count ← count + 1) < litda.darte;
   dismes(2, $ltdeml);
   RETURN(FALSE);
   END.


(aplgls) PROCEDURE(ls, newline);
   %get another lsrt entry for literal display area, allocating
   another blak line if newline is true.%
   %------------%
   LOCAL newls, blnkls;
   REF ls, newls, blnkls;
   &newls ← &ls + lsrtl;
   IF (&blnkls ← &newls + lsrtl) >= getrta(litda.darte) THEN
      BEGIN
      dismes(2, $ltdeml);
      RETURN(FALSE);
      END;
   blnkls.rtexis ← blnkls.rtllsg ← TRUE;
   blnkls.rtnew ← FALSE;
   newls.rtllsg ← FALSE;
   ls.rtnew ← FALSE;
   newls.rtnew ← TRUE;
   blnkls.rtxl ← litda.daleft;
```

```
    blnkls.rtx2 ← litda.daright;
    IF (blnkls.rty ← newls.rty + litda.davinc) > litda.dabottom THEN
        BEGIN
        dismes(2, $ltdeml);
        RETURN(FALSE);
        END;
    newls.rtbps ← newls.rtbpe ← newls.rtstid ←
        blnkls.rtbps ← blnkls.rtbpe ← blnkls.rtstid ←
        chbptr(/litda.dacsp/.L) + litda.dacsp;
    blnkls.rtfont ← litda.dafont;
    blnkls.rtcsize ← litda.dacsize;
    blnkls.rthinc ← litda.dahinc;
    IF newline THEN
        BEGIN
        newls.rtfcol ← newls.rtlcol ← blnkls.rtfcol ← blnkls.rtlcol ←
        1;
        aplsrd(&blnkls, suppress);
        BUMP litda.dacrow;
        END
    ELSE %line segment on same line%
        BEGIN
        newls.rtx1 ← ls.rtx2 ← ls.rtlcol * ls.rthinc;
        newls.rtfcol ← newls.rtlcol ← ls.rtlcol + 1;
        blnkls.rty ← newls.rty := ls.rty;
        wrtstr(&litda, &newls);
        wrtstr(&litda, &blnkls);
        END;
    RETURN(&newls);
    END.


(aplrls) PROCEDURE(ls);
    %restores lines suppressed by display of ls passed%
    %------------%
    LOCAL prvls, nxtls;
    REF prvls, nxtls, ls;
    IF (&prvls ← &ls - lsrtl) < getrta(litda.darts) THEN
        BEGIN
        litdnt(litda.dacsp);
        RETURN(FALSE);
        END;
    &nxtls ← &ls + lsrtl;
    nxtls.rtexis ← FALSE;
    ls.rtllsg ← TRUE;
    ls.rtnew ← FALSE;
    prvls.rtnew ← TRUE;
    ls.rtx1 ← litda.daleft;
    ls.rtx2 ← litda.daright;
    ls.rtfcol ← ls.rtlcol ← 1;
    ls.rtbps ← ls.rtbpe ← chbptr(/litda.dacsp/.L) + litda.dacsp;
    IF ls.rty = prvls.rty THEN
        ls.rty ← prvls.rty + litda.davinc
    ELSE
        BEGIN
        aplsrd(&ls, restore);
        BUMP DOWN litda.dacrow;
```

```
        END;
    wrtstr(&litda, &ls);
    RETURN(&prvls);
    END.


(aplsrd) PROCEDURE (ls, type);
    %suppress / restore display any line segments touched by ls; if
    TYPE is TRUE, then the display is suppressed; otherwise, it is
    restored%
    %-------------%
    LOCAL mid, dano, daadr, lsadr, ybot, ytop, lstop, dtop;
    REF daadr, lsadr, ls;
    ytop ← mid ← ls.rty + litda.datop;
    ybot ← mid + litda.davinc;
    dano ← 1;
    DO
        IF (&daadr ← dsparea(dano)) THEN
            IF (dtop ← daadr.datop) <= ybot THEN
                BEGIN
                &lsadr ← getrta(daadr.darts);
                lstop ← getrta(daadr.darte);
                DO
                    IF lsadr.rtexis AND (dtop + lsadr.rty) IN [ytop, ybot]
                    THEN
                        IF type THEN
                            BEGIN
                            rl.rhword ← lsadr.rtlsid;
                            rl.lhword ← daadr.dahandle;
                            IF NOT SKIP !JSYS ssda THEN err($ltdem2);
                            END
                        ELSE
                            BEGIN
                            rl.rhword ← lsadr.rtlsid;
                            rl.lhword ← daadr.dahandle;
                            IF NOT SKIP !JSYS rsda THEN err($ltdem2);
                            END
                UNTIL (&lsadr ← &lsadr + lsrtl) >= lstop;
                END
    UNTIL (dano ← dano + 1) > damax;
    RETURN;
    END.

FINISH of litdsp
```

MSGNLS

```
<NLS>MSGNLS.NLS;2, 29-MAY-71 11:31 WHP ;
FILE msgnls
% This silly little program is used as a timer until 10X puts
timers into the system.  it is started at location 120B, which
contains a JRST timfrk. When it is started, it assumes that
register 1 contains the number of milliseconds to wait before the
timer goes off.  When the time is up, the fork causes an interrupt
on channel 4 in its superior fork, then halts.  The fork may be
used in this way any number of times by simply setting its AC's
and restarting it at 120B. %
REGISTER r1=1, r2=2;
(timfrk) PROCEDURE;
   !JSYS 167B; %dismiss%
   r1 ← -1;
   r2 ← 2B10;
   !JSYS 132B; %cause pseudo interrupt in superior fork%
   !JSYS 170B; %halt%
   END.
FINISH of msgnls
```

```
<NIC-NLS>NCTRL.NLS;9, 11-JAN-72 22:31 BLP ;
FILE nctrl  %  L10  to  <NIC-NLS>NCTRL  %

DECLARE nofile = FALSE, noct = FALSE, notyet = 7;

%..main control..%
(wc) PROCEDURE;
     %This routine determines the command being specified and
     transfers control to the appropriate routine.%
     disarm();  an();  zap();
     CASE curchr OF
     ='a:  %append%
        qas();
     ='b:  %break%
        qbs();
     ='c:  %copy%
        BEGIN
        GROUP ← edit;  DSP(<Copy ↑*es*);
        CASE input() OF
        ='c: qcc(); %copy character%
        ='w: qcw(); %copy word%
        ='n: qcn(); %copy number%
        ='v: qcv(); %copy visible%
        ='i: qci(); %copy invisible%
        ='l: qcl(); %copy link%
        ='t: qct(); %copy text%
        ='s: qcs(); %copy statement%
        ='b: qcb(); %copy branch%
        ='p: qcp(); %copy plex%
        ='g: qcg(); %copy group%
        =CA:  REPEAT (ec)
        ENDCASE
           BEGIN
           caqm(); REPEAT;
           END;
        END;
     ='d:  %delete%
        BEGIN
        GROUP ← edit; DSP(<Delete ↑*es*);
        CASE input() OF
        ='c: qdc(); %delete character%
        ='w: qdw(); %delete word%
        ='n: qdn(); %delete number%
        ='v: qdv(); %delete visible%
        ='i: qdi(); %delete invisible%
        ='l: qdl(); %delete link%
        ='t: qdt(); %delete text%
        ='s: qds(); %delete statement%
        ='b: qdb(); %delete branch%
        ='p: qdp(); %delete plex%
        ='g: qdg(); %delete group%
        =CA: REPEAT (ec);
        ENDCASE
           BEGIN
           caqm(); REPEAT;
           END;
```

```
    END;
  ='e:  %execute%
    qex();
  ='f:  %freeze%
    qfreez();
  ='g:  %goto%
    qgoto();
  ='i:  %insert%
    BEGIN
    GROUP ← edit; DSP(<Insert ↑*es*);
    CASE input() OF
    ='c: qic(); %insert character%
    ='w: qiw(); %insert word%
    ='n: qin(); %insert number%
    ='v: qiv(); %insert visible%
    ='i: qii(); %insert invisible%
    ='l: qil(); %insert link%
    ='t: qit(); %insert text%
    ='s: qis(); %insert statement%
    ='b: qib(); %insert branch%
    ='p: qip(); %insert plex%
    ='g: qig(); %insert group%
    =CA: REPEAT (ec)
    ENDCASE
       BEGIN
       caqm(); REPEAT;
       END;
    END;
  ='j:  %jump%
    <JUMP, newjmp>();
  ='l:  %load%
    <IOCTL, ql>();
  ='m:  %move%
    BEGIN
    GROUP ← edit; DSP(<Move ↑*es*);
    CASE input() OF
    ='c: qmc(); %move character%
    ='w: qmw(); %move word%
    ='n: qmn(); %move number%
    ='v: qmv(); %move visible%
    ='i: qmi(); %move invisible%
    ='l: qml(); %move link%
    ='t: qmt(); %move text%
    ='s: qms(); %move statement%
    ='b: qmb(); %move branch%
    ='p: qmp(); %move plex%
    ='g: qmg(); %move group%
    =CA: REPEAT (ec);
    ENDCASE
       BEGIN
       caqm(); REPEAT;
       END;
    END;
  ='o:  %output%
    <IOCTL, qo>();
  ='r:  %replace%
```

```
        BEGIN
        GROUP ← edit; DSP(<Replace ↑*es*);
        CASE input() OF
        ='c: qrc(); %replace character%
        ='w: qrw(); %replace word%
        ='n: qrn(); %replace number%
        ='v: qrv(); %replace visible%
        ='i: qri(); %replace invisible%
        ='l: qrl(); %replace link%
        ='t: qrt(); %replace text%
        ='s: qrs(); %replace statement%
        ='b: qrb(); %replace branch%
        ='p: qrp(); %replace plex%
        ='g: qrg(); %replace group%
        =CA: REPEAT (ec);
        ENDCASE
           BEGIN
           caqm(); REPEAT;
           END;
        END;
    ='s:  %substitute%
        BEGIN
        DSP(<Substitute ↑ Statement);
        CASE input() OF
        ='s: qsubs(); %substitute statement%
        ='b: qsubb(); %substitute branch%
        ='p: qsubp(); %substitute plex%
        ='g: qsubg(); %substitute group%
        =CA: REPEAT ('s);
        ENDCASE
           BEGIN
           caqm(); REPEAT;
           END;
        END;
    ='t:  %transpose%
        BEGIN
        GROUP ← edit;  DSP(< Transpose ↑ *es*);
        CASE input() OF
        ='c: qtc(); %transpose character%
        ='w: qtw(); %transpose word%
        ='n: qtn(); %transpose number%
        ='v: qtv(); %transpose visible%
        ='i: qti(); %transpose invisible%
        ='l: qtl(); %transpose link%
        ='t: qtt(); %transpose text%
        ='s: qts(); %transpose statement%
        ='b: qtb(); %transpose branch%
        ='p: qtp(); %transpose plex%
        ='g: qtg(); %transpose group%
        =CA:  REPEAT (ec);
        ENDCASE
           BEGIN
           caqm(); REPEAT;
           END;
        END;
    ='u:  %update file%
```

```
      <IOCTL, quf>();
   ='v:  %view set%
      qvs();
   ='x:  %xset%
      BEGIN
      GROUP ← edit;  DSP(<Xset ↑ *es*);
      CASE input() OF
      ='c: qxc(); %set character%
      ='w: qxw(); %set word%
      ='n: qxn(); %set number%
      ='v: qxv(); %set visible%
      ='i: qxi(); %set invisible%
      ='l: qxl(); %set link%
      ='t: qxt(); %set text%
      ='s: qxs(); %set statement%
      ='b: qxb(); %set branch%
      ='p: qxp(); %set plex%
      ='g: qxg(); %set group%
      ='m: %set mode%
         BEGIN
         qxm();
         DSP(←<Xset ↑ *es*); REPEAT;
         END;
      =CA: REPEAT (ec);
      ENDCASE
         BEGIN
         caqm(); REPEAT;
         END;
      END;
   ENDCASE
      BEGIN
      caqm();
      input();
      REPEAT;
      END;
   GOTO STATE;
   END.

(wait)
   %This routine ignores all characters until it encounters:
      a command accept, in which case it returns;
      a comand delete, in which case it recreates the display
      and goes to state%
   %------------%
   PROCEDURE;
   disarm(); af();
   CASE input() OF
      =CD: GOTO STATE;
      =CA:
         BEGIN
         an(); RETURN;
         END;
      ENDCASE REPEAT;
   END.
(caqm)
   %This routine is used to display a question mark in the
```

```
   command feedback line when one of the command recognition
   routines doesn't recognize a character.  If the current
   character is anything but a command accept or commnd delete,
   the routine displays a question mark next to the command
   feedback line arrow, gets another character, and returns.%
   %------------%
   PROCEDURE;
   CASE curchr OF
      =CD: GOTO STATE;
      ENDCASE
         BEGIN
         qm();  RETURN;
         END;
   END.

(recred)
   PROCEDURE;
   LOCAL file1, file2, da, end;
   LOCAL STRING astrng[5];
   REF da;
   *astrng* ← NULL;
   dn(&astrng);
   IF NOT cdeasy THEN %regenerate display for all da's%
      alldsp()
   ELSE %set dadsp flag for appropriate da's%
      BEGIN
      IF cd1 = endfil THEN file1 ← -1
      ELSE file1 ← cd1.stfile;
      IF cd2 = endfil THEN file2 ← -1
      ELSE file2 ← cd2.stfile;
      IF file1 NOT IN [1, filcnt] THEN file1 ← -1;
      IF file2 NOT IN [1, filcnt] THEN file2 ← -1;
      end ← (&da ← $dpyarea) + da1*dacnt;
      DO
         IF da.daexis AND NOT da.daempty AND (da.dacsp.stfile =
         file1 OR da.dacsp.stfile = file2) THEN
            da.dadsp ← TRUE
      UNTIL (&da ← &da + da1) >= end;
      <DSPGEN, seldsp>();
      END;
   cdeasy ← FALSE;
   <NLS, AUXCOD, pause>(50);
      % pause for 50 milliseconds to keep tenex happy %
   RETURN;
   END.
(dstart)
   PROCEDURE;
   STATE ← dstrt,spec;
   cdeasy ← FALSE;
   recred();
   cmdrst();
   END.
(nlsrst)
   PROCEDURE;
   IF nlmode = typewriter THEN reset()
   ELSE cmdrst();
```

```
      END.
   (cmdrst)
      PROCEDURE;
      STATE ← zz,spec; DSP(<Command Reset ←);
      af();
      incse ← FALSE;
      main();
      END.
   (main)
      PROCEDURE;
      disarm(); croot(); zap();
      ON SIGNAL =-4: %called from rubout% cmdrst(); ELSE SIGNAL;
      input(); wc();
      END.
%..append..%
   (qas) %append statement command%
      PROCEDURE;
      LOCAL cdot;
      GROUP ← edit;  STATE ← xa,edit; ENTITY ← s,Statement;
      DSP(←< Append Statement);
      LOOP
         BEGIN
         *lit* ← NULL;
         cdot ← in2stca($b1, $b2, $lit);
         LOOP
            BEGIN
            xas($b1, $b2, $lit);
            recred();
            IF NOT cdot THEN EXIT;
            *lit* ← NULL;
            cdot ← in1stca($b2, $lit);
            END;
         END;
      END.
%..break..%
   (qbs)  %break statement command%
      PROCEDURE;
      LOCAL STRING levj[20];
      GROUP ← edit;  STATE ← bs, edit;  ENTITY ← s, Statement;
      DSP(←< Break Statement);
      LOOP
         BEGIN
         *lit* ← NULL;
         INPUT BUG b1 LEVADJ levj TEXT lit CA;
         xbs($b1, $levj, $lit);
         recred();
         END;
      END.
%..copy..%
   (qcc) %Copy Character command%
      PROCEDURE;
      STATE ← cc, edit; ENTITY ← c, Character;
      DSP(←< Copy Character );
      qcmt1($xcc) END.
   (qcw) %Copy Word Command%
      PROCEDURE;
```

```
   STATE ← cw, edit; ENTITY ← w, Word;
   DSP(← < Copy Word );
   qcmt1($xcw);
   END.
(qcn) %Copy Number Command%
   PROCEDURE;
   STATE ← cn, edit; ENTITY ← n, Number;
   DSP(← < Copy Number );
   qcmt1($xcn);
   END.
(qcv) %Copy Visible Command%
   PROCEDURE;
   STATE ← cv, edit; ENTITY ← v, Visible;
   DSP(← < Copy Visible );
   qcmt1($xcv);
   END.
(qci) %Copy Invisible Command%
   PROCEDURE;
   STATE ← ci, edit; ENTITY ← i, Invisible;
   DSP(← < Copy Invisible );
   qcmt1($xci);
   END.
(qcl) %Copy Link Command%
   PROCEDURE;
   STATE ← cl, edit; ENTITY ← l, Link;
   DSP(← < Copy Link );
   qcmt1($xcl);
   END.
(qcmt1) PROCEDURE(proc);
   LOCAL cdot;
   REF proc;
   LOOP
      BEGIN
      cdot ← <PRMSPC, in2ca>($b1, $b2);
      proc($b1, $b2);
      recred();
      WHILE cdot DO
         BEGIN
         cdot ← in1ca($b2);
         proc($b1, $b2);
         recred();
         END;
      END;
   END.
(qct) %Copy Text Command%
   PROCEDURE;
   STATE ← ct, edit; ENTITY ← t, Text;
   DSP(← < Copy Text );
   qcmt2($xct);
   END.
(qcmt2) PROCEDURE(proc);
   LOCAL cdot;
   REF proc;
   LOOP
      BEGIN
      cdot ← in3ca($b1, $b2, $b3);
```

```
      proc($b1, $b2, $b3);
      recred();
      WHILE cdot DO
         BEGIN
         cdot ← in2ca($b2, $b3);
         proc($b1, $b2, $b3);
         recred();
         END;
      END;
   END.
(qcs) %Copy Statement Command%
   PROCEDURE;
   STATE ← cs, edit; ENTITY ← s, Statement;
   DSP(← < Copy Statement );
   qcms1($xcs);
   END.
(qcb) %Copy Branch Command%
   PROCEDURE;
   STATE ← cb,edit; ENTITY ← b,Branch;
   DSP(← < Copy Branch );
   qcms1($xcb);
   END.
(qcp) %Copy Plex Command%
   PROCEDURE;
   STATE ← cp, edit; ENTITY ← p, Plex;
   DSP(← < Copy Plex );
   qcms1($xcp);
   END.
(qcms1) PROCEDURE(proc);
   LOCAL cdot;
   LOCAL STRING levj[20];
   REF proc;
   LOOP
      BEGIN
      cdot ← in2sadj($b1, $b2, $levj);
      b1 ← proc($b1, $b2, $levj);
      recred();
      WHILE cdot DO
         BEGIN
         cdot ← in1sadj($b2, $levj);
         b1 ← proc($b1, $b2, $levj);
         recred();
         END;
      END;
   END.
(qcg) %Copy Group Command%
   PROCEDURE;
   STATE ← cg,edit; ENTITY ← g,Group;
   DSP(← < Copy Group );
   qcms2($xcg);
   END.
(qcms2) PROCEDURE(proc);
   LOCAL cdot;
   LOCAL STRING levj[20];
   REF proc;
   LOOP
```

```
        BEGIN
        cdot ← in3sadj($b1, $b2, $b3, $levj);
        b1 ← proc($b1, $b2, $b3, $levj);
        recred();
        WHILE cdot DO
           BEGIN
           cdot ← in2sadj($b2, $b3, $levj);
           b1 ← proc($b1, $b2, $b3, $levj);
           recred();
           END;
        END;
     END.
%..delete..%
  (qdc) %Delete Character Command%
     PROCEDURE;
     STATE ← dc, edit; ENTITY ← c, Character;
     DSP(← < Delete Character);
     qdtl($xdc);
     END.
  (qdw) %Delete Word Command%
     PROCEDURE;
     STATE ← dw, edit; ENTITY ← w, Word;
     DSP(← < Delete Word);
     qdtl($xdw);
     END.
  (qdn) %Delete Number Command%
     PROCEDURE;
     STATE ← dnn, edit; ENTITY ← n, Number;
     DSP(← < Delete Number);
     qdtl($xdn);
     END.
  (qdv) %Delete Visible Command%
     PROCEDURE;
     STATE ← dv, edit; ENTITY ← v, Visible;
     DSP(← < Delete Visible);
     qdtl($xdv);
     END.
  (qdi) %Delete Invisible Command%
     PROCEDURE;
     STATE ← di,edit; ENTITY ← i,Invisible;
     DSP(← < Delete Invisible);
     qdtl($xdi);
     END.
  (qdl) %Delete Link Command%
     PROCEDURE;
     STATE ← dl, edit; ENTITY ← l, Link;
     DSP(← < Delete Link);
     qdtl($xdl);
     END.
  (qdtl) PROCEDURE(proc);
     REF proc;
     LOOP BEGIN
        inlca($b1);
        proc($b1);
        recred();
        END;
```

```
      END.
   (qdt) %Delete Text Command%
      PROCEDURE;
      STATE ← dt, edit; ENTITY ← t, Text;
      DSP(← < Delete Text);
      qdt2($xdt);
      END.
   (qdt2) PROCEDURE(proc);
      REF proc;
      LOOP BEGIN
         in2ca($b1, $b2);
         proc($b1, $b2);
         recred();
         END;
      END.
   (qds) %Delete Statement Command%
      PROCEDURE;
      STATE ← ds, edit; ENTITY ← s, Statement;
      DSP(← < Delete Statement);
      qds1($xds);
      END.
   (qdb) %Delete Branch Command%
      PROCEDURE;
      STATE ← db, edit; ENTITY ← b, Branch;
      DSP(← < Delete Branch);
      qds1($xdb) END.
   (qdp) %Delete Plex Command%
      PROCEDURE;
      STATE ← dp, edit; ENTITY ← p, Plex;
      DSP(← < Delete Plex);
      qds1($xdp) END.
   (qdg) %Delete Group Command%
      PROCEDURE;
      STATE ← dg, edit; ENTITY ← g, Group;
      DSP(← < Delete Group);
      qds2($xdg) END.
   (qds1) PROCEDURE(proc);
      REF proc;
      LOOP
         BEGIN
         in1sca($b1);
         proc($b1);
         recred();
         END;
      END.
   (qds2) PROCEDURE(proc);
      REF proc;
      LOOP
         BEGIN
         in2sca($b1, $b2);
         proc($b1, $b2);
         recred();
         END;
      END.
%..execute..%
   (qex) PROCEDURE;
```

```
LOCAL
   dir, fileno, da, jfn, filtyp, strtyp, chngflg, newmod, newdev,
   inswtch, errs, i, tabfld, dlleft, dlrght;
LOCAL TEXT POINTER stid, stid1, stid2, tp;
LOCAL STRING namstng[60], str[400], levj[20];
REF da;
DSP(<Execute ↑);
CASE input() OF
   ='a: %assimilate%
      BEGIN
      DSP (<Assimilate ↑Branch);
      CASE input() OF
         ='b:
            BEGIN
            strtyp ← brnchv;
            DSP(...Branch);
            END;
         ='g:
            BEGIN
            strtyp ← groupv;
            DSP(...Group);
            END;
         ='p:
            BEGIN
            strtyp ← plexv;
            DSP(...Plex);
            END;
         ='s:
            BEGIN
            strtyp ← stmtv;
            DSP(...Statement);
            END;
         =CA: REPEAT CASE ('b);
         ENDCASE
            BEGIN
            caqm();
            REPEAT;
            END;
      DSP (←);
      INPUT STID stid;  %get target statement%
      CASE strtyp OF
         =stmtv, =brnchv:
            BEGIN
            INPUT STID stid1;
            stid2 ← stid1;
            END;
         =groupv:
            BEGIN
            INPUT STID stid1 STID stid2;
            stid1 ← <STRMNP, grptst> (stid1, stid2 : stid2);
            END;
         =plexv:
            BEGIN
            INPUT STID stid1;
            stid1 ← <STRMNP, plxset> (stid1 : stid2);
            END;
```

```
            ENDCASE err(0);
         INPUT LEVADJ levj CA;
         invspc (&da ← lda());
         IF sysvspec.vsrlev THEN
            sysvspec ← <SEQGEN, reslev> (sysvspec, getlev (stid1));
         IF strtyp = stmtv THEN
            sysvspec.vslev ← getlev (stid1);
         xea (stid, stid1, stid2, $levj, sysvspec, sysvspec[1],
         da.dausqcod, da.dacacode);
         freflnt();
         recred();
         END;
      ='b: %Browse Mode%
         BEGIN
         DSP(... Browse ↑Enter);
         cdset(FALSE);
         CASE input() OF
            =CA, ='e:
               BEGIN
               DSP(←);
               &da ← dsparea(lcda());
               xebe(da.dacsp.stfile);
               END;
            ='l:
               BEGIN
               DSP(... Browse Leave);
               DSP(←);
               WHILE input() # CA DO caqm();
               &da ← dsparea(lcda());
               DSP(↑Keep Changes);
               CASE input() OF
                  ='y, =CA:
                     BEGIN
                     DSP(... Changes Yes);
                     chngflg ← TRUE;
                     END;
                  ='n:
                     BEGIN
                     DSP(... Changes No);
                     chngflg ← FALSE;
                     END;
                  ENDCASE
                     BEGIN
                     caqm();
                     REPEAT;
                     END;
               xebl(chngflg, da.dacsp.stfile);
               END;
            ENDCASE
               BEGIN
               caqm();
               REPEAT;
               END;
         recred();
         END;
      ='c: %Content Analyzer%
```

```
        BEGIN
        DSP(<Content Analyzer←);
        *str* ← NULL;
        INPUT (BUG tp  CA inswtch ← TRUE; /
           TEXT str CA inswtch ← FALSE;);
        IF NOT inswtch THEN
           BEGIN
           *str* ← *str*, "; ";
           FIND SF(*str*) ↑tp;
           END;
        IF (errs ← xgpc ($tp, lda())) THEN
           BEGIN
           *str* ← STRING(errs), " error(s):  Type CA.";
           dismes (1, $str);
           clrbuf (0);  % clear input buffer %
           LOOP IF input() = CA THEN EXIT;
           dismes (0);
           END;
        END;
     ='d: %Device Specification%
        BEGIN
        DSP(<Device Specification ↑);
        err(notyet);
        CASE input() OF
           = '3:
              CASE input() OF
                 = '3:
                    BEGIN
                    newdev ← tty33;
                    newmod ← typewriter;
                    <INPFBK, dn>($"33 tty");
                    END;
                 = '5:
                    BEGIN
                    newdev ← tty35;
                    newmod ← typewriter;
                    <INPFBK, dn>($"35 tty");
                    END;
                 = '7:
                    BEGIN
                    newdev ← tty37;
                    newmod ← typewriter;
                    <INPFBK, dn>($"37 tty");
                    END;
                 ENDCASE
                    BEGIN
                    qm();
                    REPEAT 2;
                    END;
           = 'D:
              BEGIN
              newdev ← display;
              newmod ← fulldisplay;
              <INPFBK, dn>($"display");
              END;
           = 'E;
```

```
            BEGIN
            newdev ← execuport;
            newmod ← typewriter;
            <INPFBK, dn>($"execuport");
            END;
         = 'T:
            BEGIN
            newdev ← titerminal;
            newmod ← typewriter;
            <INPFBK, dn>($"ti-terminal");
            END;
         = '?:
            BEGIN
            dismes(3, $"Legal devices are:");
            dismes(3, $"  33 TTY");
            dismes(3, $"  35 TTY");
            dismes(3, $"  37 TTY");
            dismes(3, $"  Execuport");
            dismes(3, $"  Texas Inst. Terminal");
            dismes(3, $"  Display");
            r1 ← 10000;
            !JSYS dismes;
            dismes(0);
            REPEAT;
            END;
         = CD: GOTO STATE;
         ENDCASE
            BEGIN
            qm();
            REPEAT;
            END;
      xed(lcda(), newdev, newmod); %execute command%
         %DOES NOT RETURN%
      END;
   ='e: %Error termination%
      BEGIN
      DSP(←<Error termination "!"); wait(); wait(); wait();
      xee(); %execute command%
      END;
   ='f: %File Verify%
      BEGIN
      LOOP
         BEGIN
         DSP(←<File Verify);
         CASE input() OF
            =CA: BEGIN
               af();
               xef(<DACIO, lcfile>(), 1);
               EXIT LOOP;
               END;
            =CD: GOTO STATE;
            ENDCASE;
         DSP(... Checksum);
         CASE input() OF
            =CA: BEGIN
               af();
```

```
                xef(<DAC10, lcfile>(), 0);
                EXIT LOOP;
                END;
            =CD: GOTO STATE;
            ENDCASE;
        END;
    GOTO STATE;
    END;
='i: %Insert Sequential%
    BEGIN
    DSP (← <Insert Sequential);
    *levj* ← NULL;
    inlsadj (stp, $levj);
    DSP (← <Tenex File);
    filtyp ← tenfil;
    LOOP
        CASE input() OF
            ='a:
                BEGIN
                DSP(←<Assembler File);
                filtyp ← macfil;
                END;
            ='t:
                BEGIN
                DSP(←<Tenex File);
                filtyp ← tenfil;
                END;
            ='9:
                BEGIN
                DSP(←<"940" File);
                filtyp ← n40fil;
                END;
            =CD: GO TO STATE;
            =CA: EXIT LOOP;
            ENDCASE caqm();
    ON SIGNAL
        = ofilerr:   % input file could not be opened %
            BEGIN
            dismes (2, sysmsg);
            clrbuf (0);   % clear the input buffer %
            GOTO eisflnm;
            END;
        ELSE;
    (eisflnm):
        DSP (<File ↑);
        *namstng* ← NULL;
        INPUT VISIBLE namstng CA;   % get file name %
        xei (tp, $levj, $namstng, filtyp);
    recred();
    END;
='j: %Journal submission%
    BEGIN
    DSP(←< Journal );
    djcontrol();
    END;
='m: %Marker Stuff%
```

```
        BEGIN
        GROUP ← spec;
        DSP(<Marker ↑ Fix);
        CASE input() OF
           ='f: %Marker Fix%
              BEGIN
              STATE ← pf, spec;
              DSP(←<Marker Fix);
              *lit* ← NULL;
              inltca($bl, $lit);
              xemf($bl, $lit); %execute command%
              recred();
              END;
           ='l: %Marker List Show%
              BEGIN
              DSP(← ... List Show);
              xeml( lcfile( wait() )); %execute command%
              wait();
              GOTO STATE;
              END;
           ='r: %Marker Release%
              BEGIN
              DSP(... Release ↑All);
              CASE input() OF
                 ='a:
                    BEGIN
                    DSP(←);
                    xemra( lcfile( wait() )); %execute
                    command%
                    recred();
                    END;
                  ='t:
                    BEGIN
                    STATE ← prt, spec;
                    DSP(←<Marker Release Text);
                    in2ca($bl, $b2);
                    xemrt($bl, $b2); %execute command%
                    recred();
                    END;
                 =CA:  REPEAT ('a);
                 ENDCASE
                    BEGIN
                    caqm();
                    REPEAT;
                    END;
              END;
           =CA: REPEAT ('f);
           ENDCASE
              BEGIN
              caqm();
              REPEAT;
              END;
        END;
      ='n: %name delimiters%
        BEGIN
        DSP(<Name delimiters ↑ Statement);
```

```
        CASE input() OF
            ='d: %name delimiters display%
                BEGIN
                DSP(← < Name delimiters Display);
                LOOP
                    BEGIN
                    INPUT STID b1;
                    <IOCTL, clist>(noct, nofile, nofile);
                    cdset(TRUE, $b1, $b1);
                    <AUXCOD, getnmdls>(b1, $str);
                    dn ($str);
                    INPUT CA;
                    <IOCTL, clupdt>();
                    recred();
                    END;
                END;
            ='s: %name delimiters statement%
                BEGIN
                DSP(← < Name delimiters Statement);
                qens1($xens);
                END;
            ='b: %name delimiters branch%
                BEGIN
                DSP(← < Name delimiters Branch);
                qens1($xenb);
                END;
            ='p: %name delimiters plex%
                BEGIN
                DSP(← < Name delimiters Plex);
                qens1($xenp);
                END;
            ='g: %name delimiters group%
                BEGIN
                DSP(← < Name delimiters Group);
                qens2($xeng);
                END;
            =CA: REPEAT ('s);
            ENDCASE
                BEGIN
                caqm();
                REPEAT;
                END;
        END;
    ='o: %Ownership of File%
        BEGIN
        DSP(←<Ownership of File);
        wait();
        xeo(lcfile()); %execute command%
        GOTO STATE;
        END;
    ='q: %Quit%
        BEGIN
        DSP(Quit); wait();
        xeq(); %execute command%
        END;
    ='r: %Reset Partial Copy%
```

```
        BEGIN
        DSP(<Reset Partial Copy);
        &da ← lda();
        fileno ← da.dacsp.stfile;
        *namstng* ← NULL;
        filnam(fileno, $namstng);
        dn($namstng);
        wait();
        DSP("Really?");
        wait();
        xer(&da); %execute command%
        recred();
        END;
    ='s: %Secondary distribution and status%
        BEGIN
        DSP(←<S);
        CASE input() OF
            ='e: %Secondary Distribution%
                BEGIN
                DSP(←< Secondary Distribution);
                DSP(↑ Document Number "--");
                CASE lookc() OF
                    =CA,=CD: GOTO STATE;
                    IN [*0,*9]:
                        BEGIN
                        DSP(?OFF);
                        ddistd();
                        END;
                    ENDCASE
                        BEGIN
                        input();
                        qm();
                        END;
                END;
            ='t: %Status%
                BEGIN
                DSP(<Status ↑);
                CASE input() OF
                    ='v:  %viewspecs%
                        BEGIN
                        DSP(Viewspecs);
                        wait();
                        DSP(←);
                        <NLS, TXCT, curvsp>(dsparea(lcda()), $lit);
                        END;
                    ='l:  %Link stack%
                        BEGIN
                        DSP(Link stack);
                        wait();
                        DSP(←);
                        <NLS, NLSEX, xesl>($lit);
                        END;
                    ='f:  %File%
                        BEGIN
                        DSP(File);
                        wait();
```

```
                    DSP(←);
                    <NLSEX, xesf>(lcfile(), $lit); %execute
                    command%
                    END;
                ENDCASE GOTO STATE;
            *lit* ← *lit*, EOL, EOL, "Type CA";
            <LITDSP, litdpy>($lit);
            wait();
            recred();
            END;
        ENDCASE
            BEGIN
            caqm();
            REPEAT CASE;
            END;
    END;
='t: %tabstop settings%
    BEGIN
    DSP(<Tabstops Set);
    &da ← lda();
    FOR i ← $datab0 UP UNTIL > $datab9 DO
        BEGIN
        tabfld ← [i];
        *namstng* ← STRING (da.tabfld / da.dahinc);
        dn ($namstng);
        CASE lookc() OF
            =CA, =BW, =BC: NULL;
            ENDCASE *namstng* ← NULL;
        INPUT NUMBER namstng CA;
        da.tabfld ← MIN (18M, cvsno($namstng) * da.dahinc);
        END;
    END;
='u: %unlock file%
    BEGIN
    DSP(<Unlock File ←);
    fileno ← lcfile();
    *namstng* ← NULL;
    filnam(fileno, $namstng);
    dn($namstng);
    wait();
    DSP("Really?");
    wait();
    <NLSEX, xeu>(fileno);
    recred();
    GOTO STATE;
    END;
='w: %Display WSI Measurements%
    BEGIN
    DSP(<Display WSI Measurements);
    CASE lookc() OF
        =CA:
            BEGIN
            input();
            *namstng* ← "Average WSI wait = ";
            typeno(mswsit/mswsic, $namstng);
            *namstng* ← *namstng*, " milliseconds";
```

```
                dismes(1, $namstng);
                wait();
                dismes(0);
                END;
             ='r:
                BEGIN
                input();
                mswsit ← mswsic ← 0;
                dismes(2, $"WSI measurements initialized.");
                END;
             ENDCASE REPEAT CASE 2;
          END;
       ENDCASE
          BEGIN
          caqm();
          REPEAT;
          END;
   GOTO STATE;
   END.

(qens1) PROCEDURE(proc);
   LOCAL STRING str1[4], strr[4];
   LOCAL da, dlleft, dlrght;
   REF proc;
   str1.L ← strr.L ← empty;
   INPUT STID b1; da ← dsparea(lcda());
   dn ($str1);
   DSP (↑OFF < Left Name Delimiter);
   INPUT (CA / WORD str1 CA);
   dlleft ←
      IF str1.L =empty OR (FIND SF(*str1*) ("NULL"/"null")) THEN 0
      ELSE *str1*[1];
   dn ($strr);
   DSP (< Right Name Delimiter);
   INPUT (CA / WORD strr CA);
   dlrght ←
      IF strr.L =empty OR (FIND SF(*strr*) ("NULL"/"null")) THEN 0
      ELSE *strr*[1];
   DSP (< "Go?");
   INPUT CA;
   proc ($b1, dlleft, dlrght, da);
   recred();
   RETURN;
   END.
(qens2) PROCEDURE(proc);
   LOCAL STRING str1[4], strr[4];
   LOCAL da, dlleft, dlrght;
   REF proc;
   str1.L ← strr.L ← empty;
   INPUT STID b1 STID b2; da ← dsparea(lcda());
   dn ($str1);
   DSP (↑OFF < Left Name Delimiter);
   INPUT (CA / WORD str1 CA);
   dlleft ←
      IF str1.L =empty OR (FIND SF(*str1*) ("NULL"/"null")) THEN 0
      ELSE *str1*[1];
```

```
        dn ($strr);
        DSP (< Right Name Delimiter);
        INPUT (CA / WORD strr CA);
        dlrght ←
           IF strr.L =empty OR (FIND SF(*strr*) ("NULL"/"null")) THEN 0
           ELSE *strr*[1];
        DSP (< "Go?");
        INPUT CA;
        proc ($bl, dlleft, dlrght, da);
        recred();
        RETURN;
        END.
%..freeze..%
    (qfreez) %Freeze Stuff%
        PROCEDURE;
        LOCAL TEXT POINTER tptr;
        LOCAL dpa, srcdpa, tgtdpa, char;
        REF srcdpa;
        GROUP ← spec;
        DSP(< Freeze ↑ Statement);
        CASE (char ← lookc()) OF
           ='s:
              BEGIN
              input();
              REPEAT CASE 1(CA);
              END;
           =CA: %Freeze Statement%
              BEGIN
              STATE ← kf,spec;
              DSP(← < Freeze Statement);
              inlsca($bl);
              &srcdpa ← dsparea(lcda());
              <IOCTL, clist>(noct, bl.stfile, nofile);
              cdset(TRUE, $bl, endfil);
              tgtdpa ← invspc(&srcdpa);
              <AUXCOD, frzstat> (tgtdpa, bl, sysvspec, sysvspec[1]);
              dspvsp(srcdpa.davspec, srcdpa.davspc2, 3);
              <IOCTL, clupdt> ();
              recred();
              GOTO STATE;
              END;
           ='r: %Release Statement%
              BEGIN
              input();
              STATE ← kr,spec;
              DSP(← < Release Statement);
              inlsca($tptr);
              tptr[1] ← 1;
              dpa ← dsparea(lcda());
              <IOCTL, clist>(noct, tptr.stfile, nofile);
              cdset(FALSE);
              <AUXCOD, relstat> (dpa, tptr);
              <IOCTL, clupdt> ();
              recred();
              GOTO STATE;
              END;
```

```
        ='a: %Release All%
            BEGIN
            input();
            DSP(←< Release All);
            dpa ← dsparea(lcda(wait()));
            tptr ← [dpa].dacsp;
            tptr[1] ← 1;
            <IOCTL, clist>(noct, tptr.stfile, nofile);
            cdset(FALSE);
            <AUXCOD, relall>(dpa);
            <IOCTL, clupdt> ();
            recred();
            END;
        ENDCASE
            BEGIN
            input();
            caqm();
            REPEAT;
            END;
    GOTO STATE;
    END.

%..goto..%
    (qgoto)
    PROCEDURE; gc flag
    LOCAL jfn, guflag;
    LOCAL STRING flnam[100];
    DSP(<Goto ↑ Display Area Control);
    CASE input() OF
        ='b: %Baseline Package%
            <BASELN, qgb> ();
        ='d: %Display area Package%
            BEGIN
            cdeasy ← FALSE;
            <DACTRL, damain>();
            END;
        ='m: %Merge%
            BEGIN
            DSP (<Merge ↑ Plex);
            CASE input() OF
                ='b: %Merge Branch%
                    BEGIN
                    DSP (← < Merge Branch);
                    qts1($xgmeb);
                    END;
                ='g: %Merge Group%
                    BEGIN
                    DSP (← < Merge Group);
                    qts2($xgmeg);
                    END;
                ='p: %Merge Plex%
                    BEGIN
                    DSP (← < Merge Plex);
                    qts1($xgmep);
                    END;
                =CA: REPEAT ('p);
```

='c: % control file %
BEGIN
DSP(<< Control File ↑Record
gcflag←TRUE;
CASE input() OF
='n: gcflag←TRUE;
='p: BEGIN gcflag←FALSE;
DSP(<<F↑Playback
=CA: gc(gcflag);
ENDCASE
BEGIN
caqm();
REPEAT;
END;

END;

```
        ENDCASE
            BEGIN
            caqm();
            REPEAT;
            END;
    END;
='p: %Programs%
    <USRPGM, qgp> ();
='s: %Sort%
    BEGIN
    ON SIGNAL
        =sorterr: BEGIN
            dismes (1,sysmsg);
            LOOP IF input() = CA THEN EXIT;
            dismes (0);
            GOTO STATE;
            END;
        ELSE;
    DSP (<Sort ↑ Plex);
    CASE input() OF
        ='b: %Sort Branch%
            BEGIN
            DSP (← < Sort Branch);
            qds1($xgsob);
            END;
        ='g: %Sort Group%
            BEGIN
            DSP (← < Sort Group);
            qds2($xgsog);
            END;
        ='p: %Sort Plex%
            BEGIN
            DSP (← < Sort Plex);
            qds1($xgsop);
            END;
        =CA: REPEAT ('p);
        ENDCASE
            BEGIN
            caqm();
            REPEAT;
            END;
    END;
='u: %Use measurements%
    BEGIN
    DSP(←<Begin Use Measurements);
    guflag ← TRUE;
    LOOP
        CASE input() OF
            ='b:
                BEGIN
                DSP(←<Begin Use Measurements);
                guflag ← TRUE;
                END;
            ='e:
                BEGIN
                DSP(←<End Use Measurements);
```

```
                    guflag ← FALSE;
                    END;
                =CA: EXIT LOOP;
                =CD: GOTO STATE;
                ENDCASE qm();
            IF guflag THEN
               IF NOT msflag THEN
                  BEGIN
                  DSP(<To File ↑);
                  *flnam* ← NULL;
                  INPUT VISIBLE flnam CA;
                  IF NOT (jfn ←lgetjfn (0, $flnam, $txtext, gtjoosf,
                  $lit)) THEN err ($lit);
                  DSP (<"Update Interval (minutes)"↑);
                  *stno* ← NULL;
                  INPUT NUMBER stno CA;
                  <AUXCOD, initms> (jfn, cvsno ($stno));
                  END
               ELSE dismes(2, $"Measurements Being Recorded")
            ELSE <AUXCOD, closms>();
            END;
         =CA: REPEAT('d);
         ENDCASE
            BEGIN
            caqm();
            REPEAT;
            END;
      GOTO STATE;
      END.
%..insert..%
   (qic) %Insert Character Command%
      PROCEDURE;
      STATE ← ic, edit; ENTITY ← c, Character;
      DSP(←< Insert Character);
      qitxt($xic) END.
   (qiw) %Insert Word Command%
      PROCEDURE;
      STATE ← iw, edit; ENTITY ← w, Word;
      DSP(←< Insert Word);
      qitxt($xiw) END.
   (qin) %Insert Number Command%
      PROCEDURE;
      STATE ← in, edit; ENTITY ← n, Number;
      DSP(←< Insert Number);
      qitxt($xin) END.
   (qiv) %Insert Visible Command%
      PROCEDURE;
      STATE ← iv, edit; ENTITY ← v, Visible;
      DSP(←< Insert Visible);
      qitxt($xiv) END.
   (qii) %Insert Invisible Command%
      PROCEDURE;
      STATE ← ii, edit; ENTITY ← i, Invisible;
      DSP(←< Insert Invisible);
      qitxt($xii) END.
   (qil) %Insert Link Command%
```

```
   PROCEDURE;
   STATE ← il, edit; ENTITY ← l, Link;
   DSP(←< Insert Link);
   qitxt($xil) END.
(qitxt) PROCEDURE(proc);
   LOCAL cdot;
   REF proc;
   LOOP BEGIN
      *lit* ← NULL;
      cdot ← inltca($bl, $lit);
      proc($bl, $lit);
      recred();
      WHILE cdot DO
         BEGIN
         *lit* ← NULL;
         inOtca($lit);
         proc($bl, $lit);
         recred();
         END;
      END;
   END.
(qit) %Insert Text Command%
   PROCEDURE;
   STATE ← it, edit; ENTITY ← t, Text;
   DSP(←< Insert Text);
   qitxt($xic) END.
(qis) %Insert Statement Command%
   PROCEDURE;
   STATE ← is, edit; ENTITY ← s, Statement;
   DSP(←< Insert Statement);
   qist() END.
(qib) %Insert Branch Command%
   PROCEDURE;
   STATE ← ib, edit; ENTITY ← b, Branch;
   DSP(←< Insert Branch);
   qist() END.
(qip) %Insert Plex Command%
   PROCEDURE;
   STATE ← ip, edit; ENTITY ← p, Plex;
   DSP(←< Insert Plex);
   qist() END.
(qig) %Insert Group Command%
   PROCEDURE;
   STATE ← ig, edit; ENTITY ← g, Group;
   DSP(←< Insert Group);
   qist() END.
(qist) PROCEDURE;
   LOCAL cdot;
   LOCAL STRING levj[20];
   LOOP BEGIN
      *lit* ← NULL;
      INPUT STID bl LEVADJ levj TEXT lit
         (CA cdot ← 0; / C. cdot ← 1);
      bl ← xis($bl, $lit, $levj);
      recred();
      WHILE cdot DO
```

```
            BEGIN
            *lit* ← NULL;
            INPUT LEVADJ levj TEXT lit
               (CA cdot ← 0; / C. cdot ← 1);
            bl ← xis($bl, $lit, $levj);
            recred();
            END;
         END;
      END.
%..move..%
   (qmc) %Move Character Command%
      PROCEDURE;
      STATE ← mc, edit; ENTITY ← c, Character;
      DSP(← < Move Character);
      qcmt1($xmc);
      END.
   (qmw) %Move Word Command%
      PROCEDURE;
      STATE ← mw, edit; ENTITY ← w, Word;
      DSP(← < Move Word);
      qcmt1($xmw);
      END.
   (qmn) %Move Number Command%
      PROCEDURE;
      STATE ← mn, edit; ENTITY ← n, Number;
      DSP(← < Move Number);
      qcmt1($xmn);
      END.
   (qmv) %Move Visible Command%
      PROCEDURE;
      STATE ← mv, edit; ENTITY ← v, Visible;
      DSP(← < Move Visible);
      qcmt1($xmv);
      END.
   (qmi) %Move Invisible Command%
      PROCEDURE;
      STATE ← mi, edit; ENTITY ← i, Invisible;
      DSP(← < Move Invisible);
      qcmt1($xmi);
      END.
   (qml) %Move Link Command%
      PROCEDURE;
      STATE ← ml, edit; ENTITY ← l, Link;
      DSP(← < Move Link);
      qcmt1($xml);
      END.
   (qmt) %Move Text Command%
      PROCEDURE;
      STATE ← mt, edit; ENTITY ← t, Text;
      DSP(← < Move Text);
      qcmt2($xmt);
      END.
   (qms) %Move Statement Command%
      PROCEDURE;
      STATE ← ms, edit; ENTITY ← s, Statement;
      DSP(← < Move Statement);
```

```
      qcms1($xms);
      END.
   (qmb) %Move Branch Command%
      PROCEDURE;
      STATE ← mb, edit; ENTITY ← b, Branch;
      DSP(← < Move Branch);
      qcms1($xmb);
      END.
   (qmp) %Move Plex Command%
      PROCEDURE;
      STATE ← mp, edit;  ENTITY ← p, Plex;
      DSP(← < Move Plex);
      qcms1($xmp);
      END.
   (qmg) %Move Group Command%
      PROCEDURE;
      STATE ← mg, edit; ENTITY ← g, Group;
      DSP(← < Move Group);
      qcms2($xmg);
      END.
%..replace..%
   (qrc) %Replace Character Command%
      PROCEDURE;
      STATE ← rc, edit; ENTITY ← c, Character;
      DSP(←<Replace Character);
      qrt1($xrc);
      END.
   (qrw) %Replace Word Command%
      PROCEDURE;
      STATE ← rw, edit; ENTITY ← w, Word;
      DSP(←<Replace Word);
      qrt1($xrw);
      END.
   (qrn) %Replace Number Command%
      PROCEDURE;
      STATE ← rn, edit; ENTITY ← n, Number;
      DSP(←< Replace Number);
      qrt1($xrn);
      END.
   (qrv) %Replace Visible Command%
      PROCEDURE;
      STATE ← rv, edit; ENTITY ← v, Visible;
      DSP(←<Replace Visible);
      qrt1($xrv);
      END.
   (qri) %Replace Invisible%
      PROCEDURE;
      STATE ← ri, edit; ENTITY ← i, Invisible;
      DSP(←<Replace Invisible);
      qrt1($xri);
      END.
   (qrl) %Replace Link Command%
      PROCEDURE;
      STATE ← rl, edit; ENTITY ← l, Link;
      DSP(←<Replace Link);
      qrt1($xrl);
```

```
   END.
(qrt) %Replace Text Command%
   PROCEDURE;
   STATE ← rt, edit; ENTITY ← t, Text;
   DSP(←<Replace Text);
   qrt2($xrt);
   END.
(qrt1) PROCEDURE(proc);
   LOCAL rpf;
   REF proc;
   LOOP
      BEGIN
      *lit* ← NULL;
      b2 ← endfil;
      b2[1] ← 1;
      INPUT BUG b1
         (BUG b2 CA rpf ← TRUE; /
         TEXT lit CA rpf ← FALSE);
      proc(rpf, $b1, $b2, $lit);
      recred();
      END;
   END.
(qrt2) PROCEDURE(proc);
   LOCAL rpf;
   REF proc;
   LOOP
      BEGIN
      *lit* ← NULL;
      b3 ← b4 ← endfil;
      b3[1] ← b4[1] ← 1;
      INPUT BUG b1 BUG b2
         (BUG b3 BUG b4 CA rpf ← TRUE; /
         TEXT lit CA rpf ← FALSE);
      proc(rpf, $b1, $b2, $b3, $b4, $lit);
      recred();
      END;
   END.
(qrs) %Replace Statement Command%
   PROCEDURE;
   STATE ← rs, edit; ENTITY ← s, Statement;
   DSP(←<Replace Statement);
   qrs1($xrs);
   END.
(qrb) %Replace Branch Command%
   PROCEDURE;
   STATE ← rb, edit; ENTITY ← b, Branch;
   DSP( ← <Replace Branch);
   qrs1($xrb);
   END.
(qrp) %Replace Plex Command%
   PROCEDURE;
   STATE ← rp, edit; ENTITY ← p, Plex;
   DSP(←<Replace Plex);
   qrs1($xrp);
   END.
(qrg) %Replace Group Command%
```

```
   PROCEDURE;
   STATE ← rg, edit; ENTITY ← g, Group;
   DSP(←<Replace Group);
   qrs2($xrg);
   END.
(qrs1) PROCEDURE(proc);
   LOCAL rpf, cdot;
   LOCAL STRING levj[20];
   REF proc;
   LOOP
      BEGIN
      *lit* ← NULL;
      b2 ← endfil;
      b2[1] ← 1;
      INPUT STID b1
         (STID b2 (C. rpf ← 3; /
            CA rpf ← 1) /
         TEXT lit (C. rpf ← 2; /
            CA rpf ← 0));
      b1 ← proc(rpf .A 1, $b1, $b2, $lit);
      recred();
      cdot ← rpf .A 2;
      WHILE cdot DO
         BEGIN
         *lit* ← NULL;
         INPUT LEVADJ levj TEXT lit
            (CA cdot ← 0; / C. cdot ← 1);
         b1 ← xis($b1, $lit, $levj);
         recred();
         END;
      END;
   END.
(qrs2) PROCEDURE(proc);
   LOCAL rpf, cdot;
   LOCAL STRING levj[20];
   REF proc;
   LOOP
      BEGIN
      *lit* ← NULL;
      b3 ← b4 ← endfil;
      b3[1] ← b4[1] ← 1;
      INPUT STID b1 STID b2
         (STID b3 STID b4 (C. rpf ← 3; / CA rpf ← 1) /
         TEXT lit (C. rpf ← 2; / CA rpf ← 0));
      b1 ← proc(rpf .A 1, $b1, $b2, $b3, $b4, $lit);
      recred();
      cdot ← rpf .A 2;
      WHILE cdot DO
         BEGIN
         *lit* ← NULL;
         INPUT LEVADJ levj TEXT lit
            (CA cdot ← 0; / C. cdot ← 1);
         b1 ← xis($b1, $lit, $levj);
         recred();
         END;
      END;
```

```
    END.
%..substitute..%
    (qsubs) %Substitute Statement%
        PROCEDURE;
        LOCAL da;
        STATE ← xs, spec;
        DSP( ←< Substitute Statement);
        inlsca($bl);
        cdset(TRUE, $bl, $bl);
        IF NOT da ← dsparea(lcda()) THEN
            err($"lcda failed, qsubb");
        <LNKSUB, subcon>(stmtv, da, bl)
        END.
    (qsubb) %Substitute Branch%
        PROCEDURE;
        LOCAL da;
        STATE ← xb, spec;
        DSP( ←< Substitute Branch);
        cdset(FALSE);
        inlsca($bl);
        IF NOT da ← dsparea(lcda()) THEN
            err($"lcda failed, qsubb");
        <LNKSUB, subcon>(brnchv, da, bl)
        END.
    (qsubp) %Substitute Plex%
        PROCEDURE;
        LOCAL da, stid1, stid2;
        STATE ← xp, spec;
        DSP( ←< Substitute Plex);
        inlsca($bl);
        stid1 ← <STRMNP, plxset>(bl : stid2);
        IF NOT da ← dsparea(lcda()) THEN
            err($"lcda failed, qsubp");
        cdset(FALSE);
        <LNKSUB, subcon>(plexv, da, stid1, stid2)
        END.
    (qsubg) %Substitute Group%
        PROCEDURE;
        LOCAL da, stid1, stid2;
        STATE ← xg, spec;
        DSP( ←< Substitute Group);
        in2sca($bl, $b2);
        stid1 ← <STRMNP, grptst>(bl, b2 : stid2);
        IF NOT da ← dsparea(lcda()) THEN
            err($"lcda failed, qsubg");
        cdset(FALSE);
        <LNKSUB,subcon>(groupv, da, stid1, stid2)
        END.
    %.....Control Substitute Code for DNLS.....%
        (subcon) PROCEDURE (type, da, stid1, stid2); %substitute control%
            LOCAL STRING dspstr[600]; %lit reg and display string for
            pairs%
            LOCAL STRING subm[30]; %string for "Subs=" message%
            LOCAL subdsp[200B];
            REF da;
            CALL sbinit($subhed, $subrec, $subdsp, $auxlit);
```

```
        *auxlit* ← NULL;
        *dspstr* ← NULL;
           %clear the lit reg and display string for collecting pairs%
        sbpair($subnum, $subhed, $dspstr);
           %collect up to 'subnum' pairs%
        substitute(type, stid1, stid2, $subhed, &da);
           %do the substitutions%
        *subm* ← "Subs: ", STRING(subcnt), ", Type CA";
        dismes(1, $subm);
           %display number of substitutions%
        LOOP IF input() = CA THEN EXIT;
           %wait for a command accept%
        dismes(0);
        recred();
           %recreate the display%
        GOTO STATE;
           %go to previous command state%
        END.

    (sbpair) PROCEDURE (count, sb, dspstr); %collect a pair of
    strings%
        LOCAL sbeg1, send1, sbeg2, send2;
        LOCAL STRING rstr[80], tstr[80];
        POINTER sb;
        REF dspstr;
        LOOP
           BEGIN
           IF (count ← count - 1) < 0 THEN RETURN;
           arm();
           DSP(<Text↑);
              sbeg1 ← dspstr.L;
              colstr(&dspstr);  %collect replacement string%
              send1 ← dspstr.L;
              *dspstr* ← *dspstr*, SP;
           DSP(<for Text↑);
              sbeg2 ← dspstr.L;
              colstr(&dspstr);  %collect test string%
              send2 ← dspstr.L;
              *dspstr* ← *dspstr*, SP;
              IF send2 = sbeg2 THEN BUMP send2; %can't substitute for
              null%
           % store parameters %
              IF sbeg1 = send1 THEN
                 *rstr* ← NULL
              ELSE
                 *rstr* ← *dspstr*[sbeg1+1 TO send1];
              *tstr* ← *dspstr*[sbeg2+1 TO send2];
              CALL sbpush($rstr, $tstr, sb);
           DSP(<"Go?");
           af(); disarm();
           CASE input() OF
              =CA, ='Y, ='y:
                 BEGIN
                 DSP(<Doing it);
                 RETURN;
                 END;
```

```
            =CD:
               GOTO STATE;
            ENDCASE NULL; %get another pattern%
         END;
      END.

   (colstr) PROCEDURE (astrng); %collect a string%
      LOCAL TEXT POINTER bug1, bug2, ptr1, ptr2;
      LOCAL bgflag;
      REF astrng;
      INPUT (BUG bug1 BUG bug2 CA bgflag ← 1; /
         TEXT astrng CA bgflag ← 0);
      IF bgflag THEN
         BEGIN
         <TXTEDT, tdr> ($bug1, $bug2, $ptr1, $ptr2);
         *astrng* ← *astrng*, ptr1 ptr2;
         litdpy(&astrng);
         END;
      RETURN;
      END.

%..transpose..%
   (qtc) %Transpose Character Command%
      PROCEDURE;
      STATE ← trc, edit; ENTITY ← c, Character;
      DSP(←< Transpose Character);
      qttl($xtc);
      END.
   (qtw) %Transpose Word Command%
      PROCEDURE;
      STATE ← trw, edit; ENTITY ← w, Word;
      DSP(←< Transpose Word);
      qttl($xtw);
      END.
   (qtn) %Transpose Number Command%
      PROCEDURE;
      STATE ← trn, edit; ENTITY ← n, Number;
      DSP(←< Transpose Number);
      qttl($xtn);
      END.
   (qtv) %Transpose Visible Command%
      PROCEDURE;
      STATE ← trv, edit; ENTITY ← v, Visible;
      DSP(←< Transpose Visible);
      qttl($xtv);
      END.
   (qti) %Transpose Invisible Command%
      PROCEDURE;
      STATE ← tri, edit; ENTITY ← i, Invisible;
      DSP(←< Transpose Invisible);
      qttl($xti);
      END.
   (qtl) %Transpose Link Command%
      PROCEDURE;
      STATE ← trl, edit; ENTITY ← l, Link;
      DSP(←< Transpose Link);
```

```
   qtt1($xt1);
   END.
(qtt) %Transpose Text Command%
   PROCEDURE;
   STATE ← trt, edit; ENTITY ← t, Text;
   DSP(←< Transpose Text);
   qtt2($xtt);
   END.
(qtt1) PROCEDURE(proc);
   REF proc;
   LOOP
      BEGIN
      in2ca($b1, $b2);
      proc($b1, $b2);
      recred();
      END;
   END.
(qtt2) PROCEDURE(proc);
   REF proc;
   LOOP
      BEGIN
      in4ca($b1, $b2, $b3, $b4);
      proc($b1, $b2, $b3, $b4);
      recred();
      END;
   END.
(qts) %Transpose Statement Command%
   PROCEDURE;
   STATE ← trs, edit; ENTITY ← s, Statement;
   DSP(← < Transpose Statement );
   qts1($xts);
   END.
(qtb) %Transpose Branch Command%
   PROCEDURE;
   STATE ← trb, edit; ENTITY ← b, Branch;
   DSP(← < Transpose Branch );
   qts1($xtb);
   END.
(qtp) %Transpose Plex Command%
   PROCEDURE;
   STATE ← trp, edit; ENTITY ← p, Plex;
   DSP(← < Transpose Plex );
   qts1($xtp);
   END.
(qtg) %Transpose Group Command%
   PROCEDURE;
   STATE ← trg, edit; ENTITY ← g, Group;
   DSP(← < Transpose Group );
   qts2($xtg);
   END.
(qts1) PROCEDURE(proc);
   REF proc;
   LOOP
      BEGIN
      in2sca($b1, $b2);
      proc($b1, $b2);
```

```
      recred();
      END;
   END.
(qts2) PROCEDURE(proc);
   REF proc;
   LOOP
      BEGIN
      in4sca($b1, $b2, $b3, $b4);
      proc($b1, $b2, $b3, $b4);
      recred();
      END;
   END.
%..view set..%
 (qvs) PROCEDURE; %view set command%
    LOCAL da;
    REF da;
    DSP(<View Set↑);
    IF &da ← <PRMSPC, invspc>(dsparea(lcda())) THEN
      BEGIN
      da.davspec ← sysvspec;
      da.davspc2 ← sysvspec[1];
      END;
    recred();
   RETURN;
    END.

%..shift case..%
(qxc) %Set Character Command%
   PROCEDURE;
   STATE ← xc, edit; ENTITY ← c, Character;
   DSP(←<Xset Character);
   qxtl($xxc);
   END.
(qxw) %Set Word Command%
   PROCEDURE;
   STATE ← xw, edit; ENTITY ← w, Word;
   DSP(←<Xset Word);
   qxtl($xxw);
   END.
(qxn) %Set Number Command%
   PROCEDURE;
   STATE ← xn, edit; ENTITY ← n, Number;
   DSP(←<Xset Number);
   qxtl($xxn);
   END.
(qxv) %Set Visible Command%
   PROCEDURE;
   STATE ← xv, edit; ENTITY ← v, Visible;
   DSP(←<Xset Visible);
   qxtl($xxv);
   END.
(qxi) %Set Invisible Command%
   PROCEDURE;
   STATE ← xi, edit; ENTITY ← i, Invisible;
   DSP(←<Xset Invisible);
   qxtl($xxi);
```

```
   END.
(qxl) %Set Link Command%
   PROCEDURE;
   STATE ← xl, edit;  ENTITY ← l, Link;
   DSP(←<Xset Link);
   qxtl($xxl);
   END.
(qxtl) PROCEDURE(proc);
   REF proc;
   LOOP BEGIN
      inlca($bl);
      proc($bl);
      recred();
      END;
   END.
(qxt2) PROCEDURE(proc);
   REF proc;
   LOOP BEGIN
      in2ca($bl, $b2);
      proc($bl, $b2);
      recred();
      END;
   END.
(qxt) %Set Text Command%
   PROCEDURE;
   STATE ← xt, edit; ENTITY ← t, Text;
   DSP(←<Xset Text);
   qxt2($xxt);
   END.
(qxs) %Set Statement Command%
   PROCEDURE;
   STATE ← xxxs, edit; ENTITY ← s, Statement;
   DSP(←<Xset Statement);
   qxsl($xxs) END.
(qxb) %Set Branch Command%
   PROCEDURE;
   STATE ← xxxb, edit; ENTITY ← b, Branch;
   DSP(←<Xset Branch);
   qxsl($xxb) END.
(qxp) %Set Plex Command%
   PROCEDURE;
   STATE ← xxxp, edit; ENTITY ← p, Plex;
   DSP(←<Xset Plex);
   qxsl($xxp) END.
(qxg) %Set Group Command%
   PROCEDURE;
   STATE ← xxxg, edit; ENTITY ← g, Group;
   DSP(←<Xset Group);
   qxs2($xxg) END.
(qxsl) PROCEDURE(proc);
   REF proc;
   LOOP BEGIN
      inlsca($bl);
      proc($bl);
      recred();
      END;
```

```
    END.
(qxs2) PROCEDURE(proc);
    REF proc;
    LOOP BEGIN
        in2sca($b1, $b2);
        proc($b1, $b2);
        recred();
        END;
    END.
(qxm) %Set Mode...%
    PROCEDURE;
    DSP(←<Xset Mode ↑);
    CASE input() OF
        ='c: %Capital%
            BEGIN
            DSP (Capital ↑);
            xsucm(); %execute command%
            REPEAT;
            END;
        ='l: %Lower%
            BEGIN
            DSP (Lower ↑);
            xslcm(); %execute command%
            REPEAT;
            END;
        =CD:
            GOTO STATE;
        =CA:
            BEGIN
            DSP (←);
            RETURN;
            END
        ENDCASE
            REPEAT;
    END.
FINISH
```

NLSEX

```
<NLS>NLSEX.NLS;58, 30-DEC-71 13:58 BLP ;
FILE nlsex  % L10  <REL-NLS>NLSEX.REL %

%Declarations%

    DECLARE
      nofile = 0, noct = 0,ctcsp = 1, ctfrz = 2,
      ctcpfz = 3, ctmkr = 8, ctcmk = 9, ctcfm = 11;
%clist utility%

   (cdset) PROCEDURE(cdezflag, ptr1, ptr2);
      IF cdeasy ← cdezflag THEN
         BEGIN
         IF ptr1 # endfil THEN
            BEGIN
            cd1 ← [ptr1]; cd1[1] ← [ptr1+1];
            END
         ELSE
            BEGIN
            cd1 ← endfil; cd1[1] ← 1;
            END;
         IF ptr2 # endfil THEN
            BEGIN
            cd2 ← [ptr2]; cd2[1] ← [ptr2+1];
            END
         ELSE
            BEGIN
            cd2 ← endfil; cd2[1] ← 1;
            END;
         END;
      RETURN END.

%append%

   (xas) %Execute Append Statement Command%
      %construct and send append statement message to core NLS%
      PROCEDURE (bug1, bug2, astrng);
      REF bug1, bug2;
      <IOCTL, clist> (ctcfm, bug1.stfile, bug2.stfile);
      cdset(FALSE);
      cas(bug1, bug2, astrng); %a call for now%
      <IOCTL, clupdt> ();
      RETURN;
      END.

%break%

   (xbs) %Execute Break Statement Command%
      %construct and send break statement message to core NLS%
      PROCEDURE ( bug, levstg, astrng);
      LOCAL newstid;
      REF bug;
      <IOCTL, clist> (ctcmk, bug.stfile, endfil);
      cdset(FALSE);
      newstid ← cbs(&bug, levstg, astrng);
      <IOCTL, clupdt> ();
```

```
      RETURN( newstid );
      END.
%copy%

   (xcmtl) PROCEDURE(proc, bugl, bug2);
      REF proc, bugl, bug2;
      <IOCTL, clist> (ctcmk, bugl.stfile, bug2.stfile);
      cdset(TRUE, &bugl, &bug2);
      proc(&bugl, &bug2, $clhead); %a call for now%
      <IOCTL, clupdt> ();
      RETURN;
      END.

   (xcc) %Execute Copy Character Command%
      %construct and send Copy Character message to core NLS%
      PROCEDURE (bugl, bug2);
      xcmtl($ccc, bugl, bug2);
      RETURN;
      END.

   (xcw) %Execute Copy Word Command%
      %construct and send Copy Word message to core NLS%
      PROCEDURE (bugl, bug2);
      xcmtl($ccw, bugl, bug2);
      RETURN;
      END.

   (xcn) %Execute Copy Number Command%
      %construct and send Copy Number message to core NLS%
      PROCEDURE (bugl, bug2);
      xcmtl($ccn, bugl, bug2);
      RETURN;
      END.

   (xcv) %Execute Copy Visible Command%
      %construct and send Copy Visible message to core NLS%
      PROCEDURE (bugl, bug2);
      xcmtl($ccv, bugl, bug2);
      RETURN;
      END.

   (xci) %Execute Copy Invisible Command%
      %construct and send Copy Invisible message to core NLS%
      PROCEDURE (bugl, bug2);
      xcmtl($cci, bugl, bug2);
      RETURN;
      END.

   (xcl) %Execute Copy Link Command%
      %construct and send Copy Link message to core NLS%
      PROCEDURE (bugl, bug2);
      xcmtl($ccl, bugl, bug2);
      RETURN;
      END.

   (xct) %Execute Copy Text Command%
```

```
   %construct and send Copy Text message to core NLS%
   PROCEDURE (bug1, bug2, bug3);
   xcmt2($corect, bug1, bug2, bug3);
   RETURN;
   END.

(xcmt2) PROCEDURE(proc, bug1, bug2, bug3);
   REF proc, bug1, bug2;
   <IOCTL, clist> (ctcmk, bug1.stfile, bug2.stfile);
   cdset(TRUE, &bug1, &bug2);
   proc(&bug1, &bug2, bug3, $c1head); %a call for now%
   <IOCTL, clupdt> ();
   RETURN;
   END.

(xcms1) PROCEDURE(proc, bug1, bug2, levstg);
   LOCAL newsid;
   REF proc, bug1, bug2;
   <IOCTL, clist> (ctcsp, bug1.stfile, bug2.stfile);
   cdset(FALSE);
   newsid ← proc(bug1, bug2, levstg); %a call for now%
   <IOCTL, clupdt> ();
   RETURN(newsid);
   END.

(xcs) %Execute Copy Statement Command%
   %construct and send Copy Statement message to core NLS%
   PROCEDURE (bug1, bug2, levstg);
   RETURN(xcms1($ccs, bug1, bug2, levstg));
   END.

(xcb) %Execute Copy Branch Command%
   %construct and send Copy Branch message to core NLS%
   PROCEDURE (bug1, bug2, levstg);
   RETURN(xcms1($ccb, bug1, bug2, levstg));
   END.

(xcp) %Execute Copy Plex Command%
   %construct and send Copy Plex message to core NLS%
   PROCEDURE (bug1, bug2, levstg);
   RETURN(xcms1($ccp, bug1, bug2, levstg));
   END.

(xcms2) PROCEDURE(proc, bug1, bug2, bug3, levstg);
   LOCAL newsid;
   REF proc, bug1, bug2, bug3;
   <IOCTL, clist> (ctcsp, bug1.stfile, bug2.stfile);
   cdset(FALSE);
   newsid ← proc(bug1, bug2, bug3, levstg);
   <IOCTL, clupdt> ();
   RETURN(newsid);
   END.

(xcg) %Execute Copy Group Command%
   %construct and send Copy Group message to core NLS%
   PROCEDURE (bug1, bug2, bug3, levstg);
```

```
    RETURN(xcms2(sccg, bug1, bug2, bug3, levstg));
    END.

%delete%

(xdxtl) PROCEDURE(proc, bug);
    REF proc, bug;
    <IOCTL, clist> (ctcmk, bug.stfile, nofile);
    cdset(TRUE, &bug, endfil);
    proc(&bug, $clhead); %a call for now%
    <IOCTL, clupdt> ();
    RETURN;
    END.

(xdc) %Execute Delete Character Command%
    %construct and send Delete Character message to core NLS%
    PROCEDURE (bug);
    xdxtl($cdc, bug);
    RETURN;
    END.

(xdw) %Execute Delete Word Command%
    %construct and send Delete Word message to core NLS%
    PROCEDURE (bug);
    xdxtl($cdw, bug);
    RETURN;
    END.

(xdn) %Execute Delete Number Command%
    %construct and send Delete Number message to core NLS%
    PROCEDURE (bug);
    xdxtl($cdn, bug);
    RETURN;
    END.

(xdv) %Execute Delete Visible Command%
    %construct and send Delete Visible message to core NLS%
    PROCEDURE (bug);
    xdxtl($cdv, bug);
    RETURN;
    END.

(xdi) %Execute Delete Invisible Command%
    %construct and send Delete Invisible message to core NLS%
    PROCEDURE (bug);
    xdxtl($cdi, bug);
    RETURN;
    END.

(xdl) %Execute Delete Link Command%
    %construct and send Delete Link message to core NLS%
    PROCEDURE (bug);
    xdxtl($cdl, bug);
    RETURN;
    END.
```

```
(xdxt2) PROCEDURE(proc, bug1, bug2);
   REF proc, bug1, bug2;
   <IOCTL, clist> (ctcmk, bug1.stfile, nofile);
   cdset(TRUE, &bug1, &bug2);
   proc(&bug1, &bug2, $clhead); %a call for now%
   <IOCTL, clupdt> ();
   RETURN;
   END.

(xdt) %Execute Delete Text Command%
   %construct and send Delete Text message to core NLS%
   PROCEDURE (bug1, bug2);
   xdxt2($cdt, bug1, bug2);
   RETURN;
   END.

(xdxs1) PROCEDURE(proc, bug);
   REF proc, bug;
   <IOCTL, clist> (ctcfm, bug.stfile, nofile);
   cdset(FALSE);
   proc(bug, $clhead); %a call for now%
   <IOCTL, clupdt> ();
   RETURN;
   END.

(xdxs2) PROCEDURE(proc, bug1, bug2);
   REF proc, bug1, bug2;
   <IOCTL, clist> (ctcfm, bug2.stfile, nofile);
   cdset(FALSE);
   proc(bug1, bug2, $clhead); %a call for now%
   <IOCTL, clupdt> ();
   RETURN;
   END.

(xds) %Execute Delete Statement Command%
   %construct and send Delete Statement message to core NLS%
   PROCEDURE (bug);
   xdxs1($cds, bug);
   RETURN;
   END.

(xdb) %Execute Delete Branch Command%
   %construct and send Delete Branch message to core NLS%
   PROCEDURE (bug);
   xdxs1($cdb, bug);
   RETURN;
   END.

(xdp) %Execute Delete Plex Command%
   %construct and send Delete Plex message to core NLS%
   PROCEDURE (bug);
   xdxs1($cdp, bug);
   RETURN;
   END.

(xdg) %Execute Delete Group Command%
```

```
      %construct and send Delete Group message to core NLS%
      PROCEDURE (bug1, bug2);
      xdxs2(&cdg, bug1, bug2);
      RETURN;
      END.

%execute commands%

   (xea) PROCEDURE   %Execute Assimilate%
      (target, src1, src2, levstg, vspec1, vspec2, usqcod, cacode);
      REF levstg;
      cdset(FALSE);
      cea (target, src1, src2, &levstg, vspec1, vspec2, usqcod, cacode);
      RETURN;
      END.

   (xebe)  %Execute Browse Enter%
      PROCEDURE (fileno);
      cebe (fileno);
      RETURN END.

   (xebl)  %Execute Browse Leave%
      PROCEDURE (lock, fileno);
      cebl (lock, fileno);
      RETURN END.

   (xed)   %Execute Device Command%
      %Construct and send Device message to core NLS%
      PROCEDURE (da, device, mode);
      REF da;
      cdset (FALSE);
      ced (&da, device, mode); %a call for now%
      RETURN;
      END.

   (xee)   %Execute Error Termination Command%
      %Construct and send Rerror message to core NLS%
      PROCEDURE;
      cee(); %a call for now%
      RETURN;
      END.

   (xef)   %Execute File Verify%
      %Construct and send File Verify message to core NLS%
      PROCEDURE (fileno, type);
      cef (fileno, type);
      RETURN;
      END.

   (xei)  %Execute Insert Sequential%
      %Construct and send Insert Sequential message to core NLS%
      PROCEDURE (stid, levstg, ifilnam, filtyp);
      REF levstg, ifilnam;
      cdset (FALSE);
      cei (stid, &levstg, &ifilnam, filtyp);
      RETURN;
```

```
   END.

(xemf)  %Execute Marker Fix Command%
   %Construct and send Marker Fix message to core NLS%
   PROCEDURE (bug, astrng);
   REF bug, astrng;
   cdset(FALSE);
   cemf (&bug, &astrng); %a call for now%
   RETURN;
   END.

(xeml)  %Execute Marker List Show Command%
   %Construct and send Marker List Show message to core NLS%
   PROCEDURE (fileno);
   cdset (TRUE, endfil, endfil);
   ceml (fileno); %a call for now%
   RETURN;
   END.

(xemra) %Execute Marker Release All Command%
   %Construct and send Marker Release All message to core NLS%
   PROCEDURE (fileno);
   cdset (FALSE);
   cemra (fileno); %a call for now%
   RETURN;
   END.

(xemrt) %Execute Marker Release Text Command%
   %Construct and send Marker Release message to core NLS%
   PROCEDURE (bug1, bug2);
   REF bug1, bug2;
   <IOCTL, clist> (ctmkr, bug1.stfile, bug2.stfile);
   cdset (TRUE, &bug1, &bug2);
   cemrt (&bug1, &bug2); %a call for now%
   <IOCTL, clupdt>();
   RETURN;
   END.

(xeo)   %Execute File Ownership Command%
   %Construct and send File Ownership message to core NLS%
   PROCEDURE (fileno);
   cdset (TRUE, endfil, endfil);
   ceo (fileno); %a call for now%
   RETURN;
   END.

(xeq)   %Execute Quit Command%
   %Construct and send Terminate message to core NLS%
   PROCEDURE;
   ceq (); %a call for now%
   RETURN;
   END.

(xer)   %Execute Reset File Command%
   %Construct and send Reset File message to core NLS%
   PROCEDURE (da);
```

```
      REF da;
      cdset (FALSE);
      <IOCTL, clist> (ctcfm, da.dacsp.stfile, nofile);
      cer (da.dacsp.stfile); %a call for now%
      da.dacsp.stpsid ← origin;
      <IOCTL, clupdt> ();
      RETURN;
      END.

   (xesf)   %Execute File Status Command%
      %Construct and send Status message to core NLS%
      PROCEDURE (fileno, astrng);
      REF astrng;
      cdset (TRUE, endfil, endfil);
      cesf (fileno, &astrng);
      RETURN;
      END.

   (xesl)   %Execute Link Stack Status Command%
      %Construct and send Status message to core NLS%
      PROCEDURE (astrng);
      REF astrng;
      cdset (TRUE, endfil, endfil);
      cesl (&astrng); %a call for now%
      RETURN;
      END.

   (xeu)   %Execute Unlock file Command%
      PROCEDURE (fileno);
      <IOCTL, clist> (ctcfm, fileno, nofile);
      cdset (FALSE);
      ceu (fileno);
      <IOEXEC, unlkclist> (); %check clist items%
      <IOCTL, clupdt> ();
      RETURN;
      END.

%goto commands%

   % Goto Baseline commands %

      (xgbe)  % Xecute Goto Baseline Enter task %
         PROCEDURE;
         cdset (FALSE);
         cgbe ();
         RETURN;
         END.

      (xgbic) % Xecute Goto Baseline Institute Conan program %
         PROCEDURE (da);
         REF da;
         cgbic (&da);
         RETURN;
         END.

      (xgbis) % Xecute Goto Baseline Institute Seqgen program%
```

```
        PROCEDURE (da);
        REF da;
        cgbis (&da);
        RETURN;
        END.

    (xgbpb) % Xecute Goto Baseline Parameter set for a Branch %
        PROCEDURE (stid, da);
        REF da;
        cgbpb (stid, &da);
        RETURN;
        END.

    (xgbps) % Xecute Goto Baseline Parameter set for a String %
        PROCEDURE (tp);
        REF tp;
        cgbps (&tp);
        RETURN;
        END.

    (xgbs)  % Xecute Goto Baseline Status of parameters%
        PROCEDURE (str);
        REF str;
        cgbs (&str);
        RETURN;
        END.

% Goto Merge commands %

    (xgmeb)  %Execute Merge Branch Command%
        PROCEDURE(bug1,bug2);
        REF bug1,bug2;
        cdset(FALSE);
        cgmeb (bug1,bug2);
        RETURN; END.

    (xgmeg)  %Execute Merge Group Command%
        PROCEDURE(bug1,bug2,bug3,bug4);
        REF bug1,bug2,bug3,bug4;
        cdset(FALSE);
        cgmeg (bug1,bug2,bug3,bug4);
        RETURN; END.

    (xgmep)  %Execute Merge Plex Command%
        PROCEDURE(bug1,bug2);
        REF bug1,bug2;
        cdset(FALSE);
        cgmep (bug1,bug2);
        RETURN; END.

% Goto user Program commands %

    (xgpc) % Xecute Content analyzer pattern compile %
        PROCEDURE (tp, da);
        REF tp, da;
        RETURN (cgpc (&tp, &da));
```

```
      END.

   (xgpd) % Xecute Deinstitute a user program %
      PROCEDURE (pgmnam);
      REF pgmnam;
      cgpd (&pgmnam);
      RETURN;
      END.

   (xgpe) % Xecute Execute a user program %
      PROCEDURE (pgmnam);
      REF pgmnam;
      cgpe (&pgmnam);
      RETURN;
      END.

   (xgpi) % Xecute Institute a user program %
      PROCEDURE (pgmnam, da, pgmfield);
      REF pgmnam, da;
      cgpi (&pgmnam, &da, pgmfield);
      RETURN;
      END.

   (xgpl) % Xecute L10 user program compile %
      PROCEDURE (stid, da);
      REF da;
      cgpl (stid, &da);
      RETURN;
      END.

   (xgpp) % Xecute Pop a user program off stack %
      PROCEDURE;
      cgpp ();
      RETURN;
      END.

   (xgpr) % Xecute Reset user program stack %
      PROCEDURE;
      cgpr ();
      RETURN;
      END.

   (xgps) % Xecute Status user programs display %
      PROCEDURE (string, da);
      REF string, da;
      cgps (&string, &da);
      RETURN;
      END.

% Goto Sort commands %

   (xgsob)  %Execute Sort Branch Command%
      PROCEDURE (bug1);
      REF bug1;
      cdset (FALSE);
      cgsob (bug1);
```

```
      RETURN; END.

   (xgsog)  %Execute Sort Group Command%
      PROCEDURE (bug1, bug2);
      REF bug1, bug2;
      cdset (FALSE);
      cgsog (bug1, bug2);
      RETURN; END.

   (xgsop)  %Execute Sort Plex Command%
      PROCEDURE (bug1);
      REF bug1;
      cdset (FALSE);
      cgsop (bug1);
      RETURN; END.

%insert%

   (xitl) PROCEDURE (proc, bug, astrng);
      REF proc, bug;
      <IOCTL, clist> (ctcmk, bug.stfile, nofile);
      cdset(TRUE, &bug, endfil);
      proc(&bug, astrng, $clhead); %a call for now%
      <IOCTL, clupdt> ();
      RETURN;
      END.

   (xic) %Execute Insert Character/Text Command%
      %construct and send Insert Character Message to core NLS%
      PROCEDURE (bug, astrng);
      xitl($cic, bug, astrng);
      RETURN;
      END.

   (xiw) %Execute Insert Word Command%
      %construct and send Insert Word Message to core NLS%
      PROCEDURE (bug, astrng);
      xitl($ciw, bug, astrng);
      RETURN;
      END.

   (xin) %Execute Insert Number Command%
      %construct and send Insert Number Message to core NLS%
      PROCEDURE (bug, astrng);
      xitl($cin, bug, astrng);
      RETURN;
      END.

   (xiv) %Execute Insert Visible Command%
      %construct and send Insert Visible Message to core NLS%
      PROCEDURE (bug, astrng);
      xitl($civ, bug, astrng);
      RETURN;
      END.

   (xii) %Execute Insert Invisible Command%
```

```
      %construct and send Insert Invisible Message to core NLS%
      PROCEDURE (bug, astrng);
      xitl($cii, bug, astrng);
      RETURN;
      END.

   (xil) %Execute Insert Link Command%
      %construct and send Insert Link Message to core NLS%
      PROCEDURE (bug, astrng);
      xitl($cil, bug, astrng);
      RETURN;
      END.

   (xit) %Execute Insert Text Command%
      %construct and send Insert Text Message to core NLS%
      PROCEDURE (bug, astrng);
      xitl($cic, bug, astrng);
      RETURN;
      END.

   (xis) %Execute Insert Statement Command%
      %construct and send Insert Statement Message to core NLS%
      PROCEDURE (bug, astrng, levstg);
      LOCAL newsid;
      REF bug;
      <IOCTL, clist> (ctcsp, bug.stfile, nofile);
      cdset(FALSE);
      newsid ← cis(bug, astrng, levstg, $clhead); %a call for now%
      <IOCTL, clupdt> ();
      RETURN(newsid);
      END.

%load%

   (xlf)   % Xecute Load File %
      PROCEDURE (flnam, da);
      LOCAL fileno;
      REF flnam, da;
      IF (fileno ← clf(&flnam)) THEN
         BEGIN
         <JUMP, pushjs> (&da);   %push link stack%
         da.dacsp ← orgstid;
         da.dacsp.stfile ← fileno;
         %push link or jump stack again%
            <JUMP, pushls> (&da, &flnam);
         END;
      RETURN;
      END.

   (xllf)   % Xecute Load Locked File %
      PROCEDURE (flnam, da);
      LOCAL fileno;
      REF flnam, da;
      IF (fileno ← cllf(&flnam)) THEN
         BEGIN
         <JUMP, pushjs> (&da);   %push link stack%
```

```
        da.dacsp. orgstid ← fileno;
        da.dacsp ←stfile;
        %push link or jump stack again%
           <JUMP, pushls> (&da, &flnam);
        END;
     RETURN;
     END.

%move%

   (xmc) %Execute Move Character Command%
      %construct and send Move Character message to core NLS%
      PROCEDURE (bug1, bug2);
      xcmt1($cmc, bug1, bug2);
      RETURN;
      END.

   (xmw) %Execute Move Word Command%
      %construct and send Move Word message to core NLS%
      PROCEDURE (bug1, bug2);
      xcmt1($cmw, bug1, bug2);
      RETURN;
      END.

   (xmn) %Execute Move Number Command%
      %construct and send Move Number message to core NLS%
      PROCEDURE (bug1, bug2);
      xcmt1($cmn, bug1, bug2);
      RETURN;
      END.

   (xmv) %Execute Move Visible Command%
      %construct and send Move Visible message to core NLS%
      PROCEDURE (bug1, bug2);
      xcmt1($cmv, bug1, bug2);
      RETURN;
      END.

   (xmi) %Execute Move Invisible Command%
      %construct and send Move Invisible message to core NLS%
      PROCEDURE (bug1, bug2);
      xcmt1($cmi, bug1, bug2);
      RETURN;
      END.

   (xml) %Execute Move Link Command%
      %construct and send Move Link message to core NLS%
      PROCEDURE (bug1, bug2);
      xcmt1($cml, bug1, bug2);
      RETURN;
      END.

   (xmt) %Execute Move Text Command%
      %construct and send Move Text message to core NLS%
      PROCEDURE (bug1, bug2, bug3);
      xcmt2($cmt, bug1, bug2, bug3);
```

```
      RETURN;
      END.

   (xms) %Execute Move Statement Command%
      %construct and send Move Statement message to core NLS%
      PROCEDURE (bug1, bug2, levstg);
      RETURN(xcms1($cms, bug1, bug2, levstg));
      END.

   (xmb) %Execute Move Branch Command%
      %construct and send Move Branch message to core NLS%
      PROCEDURE (bug1, bug2, levstg);
      RETURN(xcms1($cmb, bug1, bug2, levstg));
      END.

   (xmp) %Execute Move Plex Command%
      %construct and send Move Plex message to core NLS%
      PROCEDURE (bug1, bug2, levstg);
      RETURN(xcms1($cmp, bug1, bug2, levstg));
      END.

   (xmg) %Execute Move Group Command%
      %construct and send Move Group message to core NLS%
      PROCEDURE (bug1, bug2, bug3, levstg);
      RETURN(xcms2($cmg, bug1, bug2, bug3, levstg));
      END.

%name delimiter commands%

   (xns) %Execute Name Delimiter Statement Command%
      %construct and send Name Delimiter Statement message to core NLS%
      PROCEDURE (bug, dlleft, dlrght);
      REF bug;
      cdset (FALSE);
      cns(bug, dlleft, dlrght); %a call for now%
      RETURN;
      END.

   (xnb) %Execute Name Delimiter Branch Command%
      %construct and send Name Delimiter Branch message to core NLS%
      PROCEDURE (bug, dlleft, dlrght, da);
      REF bug;
      cdset (FALSE);
      cnb(bug, dlleft, dlrght, da); %a call for now%
      RETURN;
      END.

   (xnp) %Execute Name Delimiter Plex Command%
      %construct and send Name Delimiter Plex message to core NLS%
      PROCEDURE (bug, dlleft, dlrght, da);
      REF bug;
      cdset (FALSE);
      cnp(bug, dlleft, dlrght, da); %a call for now%
      RETURN;
      END.
```

```
    (xng) %Execute Name Delimiter Group Command%
       %construct and send Name Delimiter Group message to core NLS%
       PROCEDURE (bug1, bug2, dlleft, dlrght, da);
       REF bug1, bug2;
       cdset (FALSE);
       cng(bug1, bug2, dlleft, dlrght, da); %a call for now%
       RETURN;
       END.

%output%

    (xoa) % Xecute Output to Assembler file command %
       PROCEDURE (ofilnam, da);
       REF ofilnam, da;
       coa (&ofilnam, &da);
       RETURN;
       END.

    (xoc) % Xecute Output Compiler command %
       PROCEDURE (prcnam, ofilnam, da);
       LOCAL errors, size;
       REF prcnam, ofilnam, da;
       errors ← coc (&prcnam, &ofilnam, &da : size);
       RETURN (errors, size);
       END.

    (xod) % Xecute Output Device command %
       PROCEDURE (ofilnam, da, device);
       REF ofilnam, da;
       cod (&ofilnam, &da, device);
       RETURN;
       END.

    (xof) % Xecute Output nls File %
       PROCEDURE (newfilnam, oldfilno);
       REF newfilnam;
       cdset (FALSE);
       cof (&newfilnam, oldfilno);
       RETURN;
       END.

    (xoq) % Xecute Output Quickprint command %
       PROCEDURE (ofilnam, da);
       REF ofilnam, da;
       coq (&ofilnam, &da);
       RETURN;
       END.

    (xos) % Xecute Output Sequential command %
       PROCEDURE (ofilnam, da, forceup);
       REF ofilnam, da;
       cos (&ofilnam, &da, forceup);
       RETURN;
       END.

%replace%
```

```
(xrtl) PROCEDURE(proc, type, bug1, bug2, astrng);
   REF proc, bug1, bug2;
   <IOCTL, clist> (ctcmk, bug1.stfile, bug2.stfile);
   cdset(TRUE, &bug1, &bug2);
   proc(type, &bug1, &bug2, astrng, $clhead); %a call for now%
   <IOCTL, clupdt> ();
   RETURN;
   END.

(xrc) %Execute Replace Character Command%
   %construct and send Replace Character message to core NLS%
   PROCEDURE (type, bug1, bug2, astrng);
   xrtl($crc, type, bug1, bug2, astrng);
   RETURN;
   END.

(xrw) %Execute Replace Word Command%
   %construct and send Replace Word message to core NLS%
   PROCEDURE (type, bug1, bug2, astrng);
   xrtl($crw, type, bug1, bug2, astrng);
   RETURN;
   END.

(xrn) %Execute Replace Number Command%
   %construct and send Replace Number message to core NLS%
   PROCEDURE (type, bug1, bug2, astrng);
   xrtl($crn, type, bug1, bug2, astrng);
   RETURN;
   END.

(xrv) %Execute Replace Visible Command%
   %construct and send Replace Visible message to core NLS%
   PROCEDURE (type, bug1, bug2, astrng);
   xrtl($crv, type, bug1, bug2, astrng);
   RETURN;
   END.

(xri) %Execute Replace Invisible Command%
   %construct and send Replace Invisible message to core NLS%
   PROCEDURE (type, bug1, bug2, astrng);
   xrtl($cri, type, bug1, bug2, astrng);
   RETURN;
   END.

(xrl) %Execute Replace Link Command%
   %construct and send Replace Link message to core NLS%
   PROCEDURE (type, bug1, bug2, astrng);
   xrtl($crl, type, bug1, bug2, astrng);
   RETURN;
   END.

(xrt) %Execute Replace Text Command%
   %construct and send Replace Text message to core NLS%
   PROCEDURE (type, bug1, bug2, bug3, bug4, astrng);
   REF bug1, bug2, bug3, bug4;
```

```
      <IOCTL, clist> (ctcmk, bug1.stfile, bug3.stfile);
      cdset(TRUE, &bug1, &bug3);
      crt(type, &bug1, &bug2, &bug3, &bug4, astrng, $clhead);
      <IOCTL, clupdt> ();
      RETURN;
      END.

   (xrsl) PROCEDURE(proc, type, bug1, bug2, astrng);
      LOCAL newsid;
      REF proc, bug1, bug2;
      <IOCTL, clist> (ctcfm, bug1.stfile, bug2.stfile);
      cdset(FALSE);
      newsid ← proc(type, bug1, bug2, astrng, $clhead); %call now%
      <IOCTL, clupdt> ();
      RETURN(newsid);
      END.

   (xrs) %Execute Replace Statement Command%
      %construct and send Replace Statement message to core NLS%
      PROCEDURE (type, bug1, bug2, astrng);
      RETURN(xrsl($crs, type, bug1, bug2, astrng));
      END.

   (xrb) %Execute Replace Branch Command%
      %construct and send Replace Branch message to core NLS%
      PROCEDURE (type, bug1, bug2, astrng);
      RETURN(xrsl($crb, type, bug1, bug2, astrng));
      END.

   (xrp) %Execute Replace Plex Command%
      %construct and send Replace Plex message to core NLS%
      PROCEDURE (type, bug1, bug2, astrng);
      RETURN(xrsl($crp, type, bug1, bug2, astrng));
      END.

   (xrg) %Execute Replace Group Command%
      %construct and send Replace Group message to core NLS%
      PROCEDURE (type, bug1, bug2, bug3, bug4, astrng);
      LOCAL newsid;
      REF bug1, bug2, bug3, bug4;
      <IOCTL, clist> (ctcfm, bug1.stfile, bug3.stfile);
      cdset(FALSE);
      newsid ← crg(type, bug1, bug2, bug3, bug4,
         astrng, $clhead); %a call for now%
      <IOCTL, clupdt> ();
      RETURN(newsid);
      END.

%substitute%

   (xsubs)   % Xecute substitute statement %
      PROCEDURE (stid, sbheadb, daadd);
      %construct and send substitute statement message to
      CORE-NLS%
      cdeasy ← TRUE;
      clist(0, stid, endfil, 0, 0);
```

```
      csubs(stid, sbheadb, daadd);
      RETURN;
      END.

   (xsubb)   % Xecute substitute branch %
      PROCEDURE (stid, sbheadb, daadd);
      %construct and send substitute branch message to CORE-NLS%
      cdeasy ← FALSE;
      csubb(stid, sbheadb, daadd);
      RETURN;
      END.

   (xsubp)   % Xecute substitute plex %
      PROCEDURE (stid, sbheadb, daadd);
      %construct and send substitute plex message to CORE-NLS%
      %---------%
      cdeasy ← FALSE;
      csubp(stid, sbheadb, daadd);
      RETURN;
      END.

   (xsubg)   % Xecute substitute group %
      PROCEDURE (stid1, stid2, sbheadb, daadd);
      %construct and send substitute group message to CORE-NLS%
      %---------%
      cdeasy ← FALSE;
      csubg(stid1, stid2, sbheadb, daadd);
      RETURN;
      END.

%transpose%

   (xttl) PROCEDURE(proc, bug1, bug2);
      REF proc, bug1, bug2;
      <IOCTL, clist> (ctcmk, bug1.stfile, bug2.stfile);
      cdset(TRUE, &bug1, &bug2);
      proc(&bug1, &bug2, $clhead); %a call for now%
      <IOCTL, clupdt> ();
      RETURN;
      END.

   (xtc) %Execute Transpose Character Command%
      %construct and send Transpose Character message to core
      NLS%
      PROCEDURE (bug1, bug2);
      xttl($ctc, bug1, bug2);
      RETURN;
      END.

   (xtw) %Execute Transpose Word Command%
      %construct and send Transpose Word message to core NLS%
      PROCEDURE (bug1, bug2);
      xttl($ctw, bug1, bug2);
      RETURN;
      END.
```

```
(xtn) %Execute Transpose Number Command%
   %construct and send Transpose Number message to core NLS%
   PROCEDURE (bug1, bug2);
   xtt1($ctn, bug1, bug2);
   RETURN;
   END.

(xtv) %Execute Transpose Visible Command%
   %construct and send Transpose Visible message to core NLS%
   PROCEDURE (bug1, bug2);
   xtt1($ctv, bug1, bug2);
   RETURN;
   END.

(xti) %Execute Transpose Invisible Command%
   %construct and send Transpose Invisible message to core
   NLS%
   PROCEDURE (bug1, bug2);
   xtt1($cti, bug1, bug2);
   RETURN;
   END.

(xtl) %Execute Transpose Link Command%
   %construct and send Transpose Link message to core NLS%
   PROCEDURE (bug1, bug2);
   xtt1($ctl, bug1, bug2);
   RETURN;
   END.

(xtt) %Execute Transpose Text Command%
   %construct and send Transpose Text message to core NLS%
   PROCEDURE(bug1, bug2, bug3, bug4);
   REF bug1, bug2, bug3, bug4;
   <IOCTL, clist> (ctcmk, bug1.stfile, bug3.stfile);
   cdset(TRUE, &bug1, &bug3);
   ctt(&bug1, &bug2, &bug3, &bug4, $clhead); %a call for now%
   <IOCTL, clupdt> ();
   RETURN;
   END.

(xtsl) PROCEDURE(proc, bug1, bug2);
   REF proc, bug1, bug2;
   <IOCTL, clist> (ctcfm, bug1.stfile, bug2.stfile);
   cdset(FALSE);
   proc(bug1, bug2, $clhead); %a call for now%
   <IOCTL, clupdt> ();
   RETURN;
   END.

(xts) %Execute Transpose Statement Command%
   %construct and send Transpose Statement message to core
   NLS%
   PROCEDURE (bug1, bug2);
   xtsl($cts, bug1, bug2);
   RETURN;
   END.
```

```
    (xtb) %Execute Transpose Branch Command%
       %construct and send Transpose Branch message to core NLS%
       PROCEDURE (bug1, bug2);
       xts1($ctb, bug1, bug2);
       RETURN;
       END.

    (xtp) %Execute Transpose Plex Command%
       %construct and send Transpose Plex message to core NLS%
       PROCEDURE (bug1, bug2);
       xts1($ctp, bug1, bug2);
       RETURN;
       END.

    (xtg) %Execute Transpose Group Command%
       %construct and send Transpose Group message to core NLS%
       PROCEDURE (bug1, bug2, bug3, bug4);
       REF bug1, bug2, bug3, bug4;
       <IOCTL, clist> (ctcfm, bug1.stfile, bug3.stfile);
       cdset(FALSE);
       ctg(bug1, bug2, bug3, bug4, $c1head); %a call for now%
       <IOCTL, clupdt> ();
       RETURN;
       END.

%update file%

    (xuf)   % Xecute Update File to new or old version %
       PROCEDURE (fileno, newversion);
       RETURN (cuf (fileno, newversion));
       END.

%xset%

    (xxc) %Execute Set Character Command%
       %construct and send Set Character message to cre NLS%
       PROCEDURE (bug);
       xdxt1($cxc, bug);
       RETURN;
       END.

    (xxw) %Execute Set Word Command%
       %construct and send Set Word message to cre NLS%
       PROCEDURE (bug);
       xdxt1($cxw, bug);
       RETURN;
       END.

    (xxn) %Execute Set Number Command%
       %construct and send Set Number message to cre NLS%
       PROCEDURE (bug);
       xdxt1($cxn, bug);
       RETURN;
       END.
```

```
(xxv) %Execute Set Visible Command%
   %construct and send Set Visible message to cre NLS%
   PROCEDURE (bug);
   xdxt1($cxv, bug);
   RETURN;
   END.

(xxi) %Execute Set Invisible Command%
   %construct and send Set Invisible message to cre NLS%
   PROCEDURE (bug);
   xdxt1($cxi, bug);
   RETURN;
   END.

(xxl) %Execute Set Link Command%
   %construct and send Set Link message to cre NLS%
   PROCEDURE (bug);
   xdxt1($cxl, bug);
   RETURN;
   END.

(xxt) %Execute Set Text Command%
   %construct and send Set Text message to core NLS%
   PROCEDURE (bug1, bug2);
   xdxt2($cxt, bug1, bug2);
   RETURN;
   END.

(xxs) %Execute Set Statement Command%
   %construct and send Set Statement message to core NLS%
   PROCEDURE (bug);
   xdxs1($cxs, bug);
   RETURN;
   END.

(xxb) %Execute Set Branch Command%
   %construct and send Set Branch message to core NLS%
   PROCEDURE (bug);
   xdxs1($cxb, bug);
   RETURN;
   END.

(xxp) %Execute Set Plex Command%
   %construct and send Set Plex message to core NLS%
   PROCEDURE (bug);
   xdxs1($cxp, bug);
   RETURN;
   END.

(xxg) %Execute Set Statement Command%
   %construct and send Set Statement message to core NLS%
   PROCEDURE (bug1, bug2);
   xdxs2($cxg, bug1, bug2);
   RETURN;
   END.
```

```
%mode setting%

   (xsucm) %Execute Set Upper Case Mode Command%
      %construct and send Set Upper Case Mode message to core
      NLS%
      PROCEDURE;
      csucm(); %a call for now%
      RETURN;
      END.

   (xslcm) %Execute Set Lower Case Mode Command%
      %construct and send Set Lower Case Mode message to core
      NLS%
      PROCEDURE;
      cslcm(); %a call for now%
      RETURN;
      END.

%core display calls%

   (xdafrmt) PROCEDURE (da, wa, sa);
      %construct and send 'display area format' message to
      Core-NLS%
      %--------------%
      cdafrmt(da, wa, sa);
         %just a call for now%
      RETURN;
      END.

   (xdaupdate) PROCEDURE (da, stid1, stid2, wa, sa);
      %construct and send 'display area update' message to
      Core-NLS%
      %--------------%
      cdaupdate(da, stid1, stid2, wa, sa);
         %just a call for now%
      RETURN;
      END.

FINISH of nlsex
 &
```

PRMSPC

```
<NLS>PRMSPC.NLS;23, 16-DEC-71 16:44 MSC ; ["INPUT"];
FILE prmspc  %  L10  <REL-NLS>PRMSPC  %

%variable declarations%
   DECLARE
      name=1, word=2, contnt=3, sid=4, gotchr=0, getchr=1,
      frstchr=1;

   DECLARE bullshit; % for META bug?? %
   REGISTER a1 = 12, m = 10;

%INPUT support routines%

   (inbug) PROCEDURE(bg);
      an();
      arm();
      CASE lookc() OF
         =CD:
            BEGIN
            input();
            GOTO STATE;
            END;
         =CA, =C.:
            BEGIN
            input();
            strbug(bg);
            RETURN ;
            END;
         =BC:
            BEGIN
            input();
            BUMP [m];
            RETURN ;
            END;
         ENDCASE
            BEGIN
            BUMP [m], [m];
            RETURN;
            END;
      END.

   (instid) PROCEDURE(bg);
      LOCAL cnt;
      an();
      arm();
      CASE lookc() OF
         =CD:
            BEGIN
            input();
            GOTO STATE;
            END;
         =BUG, =C.:
            BEGIN
            input();
            strbug(bg);
            RETURN ;
```

```
        END;
    =BC:
        BEGIN
        input();
        BUMP [m];
        RETURN ;
        END;
    =SP:
        BEGIN
        input();
        *stn* ← NULL;
        cnt ← 0;
        [bg] ← origin;
        [bg].stfile ← lcfile();
        inname($stn);  %can return +1, +2, +3 past call%
        BUMP cnt, cnt;
        CASE cnt OF
            =2:
                BEGIN
                input();
                specreg($stn, name, bg);
                IF [bg] = endfil THEN
                    err($"No such statement encountered");
                RETURN;
                END;
            =1:
                BEGIN
                BUMP [m];
                RETURN;
                END;
            ENDCASE
                BEGIN
                BUMP [m],[m];
                RETURN;
                END;
        END;
    ENDCASE
        BEGIN
        BUMP [m], [m];
        RETURN;
        END;
END.

(intext)
    %This routine reads literal input from a work station,
    appends it to an A-string, and displays it in the literal
    register.  The argument passed this routine should be the
    address of the A-string to which the literal input is to
    be appended. (Note that this routine does not clear the
    A-string before it begins reading characters.)  TEXT = arbitrary
    number of characters up  to but not including a CA or Center dot.%
    %---------------%
    PROCEDURE(astrng);
    REF astrng;
    af();
    disarm();
```

```
   LOOP
      BEGIN
      CASE lookc() OF
         =CD:
            BEGIN
            input();
            GOTO STATE;
            END;
         =CA, =C.:
            RETURN;
         =BC:
            BEGIN
            input();
            IF astrng.L = empty THEN
               BEGIN
               BUMP [m];
               RETURN;
               END
            ELSE <INPFBK, bkc>(&astrng);
            END;
         =BW:
            BEGIN
            input();
            IF astrng.L = empty THEN
               BEGIN
               BUMP [m];
               RETURN;
               END
            ELSE <INPFBK, bkw>(&astrng);
            END;
         ENDCASE *astrng* ← *astrng*, input();
      litdpy(&astrng);
      END;
   END.

(innum)
   %This routine reads digits from the work station, appends
   them to an A-string, and displays them in the name register
    The argument is the address of the register
   into which the A-string is to be put. (Note that this
   routine does not clear the A-string before appending
   characters to it.).%
   %---------------%
   PROCEDURE(astrng);
   LOCAL char;
   REF astrng;
   af();
   disarm();
   LOOP
      BEGIN
      dn(&astrng);
      CASE char ← lookc() OF
         = '-:
            IF astrng.L = empty THEN
               *astrng* ← *astrng*, char
            ELSE
```

```
                BEGIN
                BUMP [m], [m];
                RETURN;
                END;
            =CA, =C.:
                BEGIN
                IF astrng.L = empty THEN
                    BEGIN
                    input();
                    strbug($b1);
                    <TXTEDT, ndr>($b1, $p1, $p2);
                    *astrng* ← p1 p2;
                    END;
                RETURN;
                END;
            =CD:
                BEGIN
                input();
                an();
                GOTO STATE;
                END;
            =D: *astrng* ← *astrng*, char;
            =BC:
                IF astrng.L = empty THEN
                    BEGIN
                    input();
                    BUMP [m];
                    RETURN;
                    END
                ELSE <INPFBK, bkc>(&astrng);
            =BW:
                IF astrng.L = empty THEN
                    BEGIN
                    input();
                    BUMP [m];
                    RETURN;
                    END
                ELSE <INPFBK, bkw>(&astrng);
            ENDCASE
                BEGIN
                BUMP [m], [m];
                RETURN;
                END;
        input();
        END;
    END.

(insr)
    %This routine reads characters from the work station,
    appends them to an A-string, and displays them in the name
    register.  The argument should be the address of the
    register into which the A-string is to be put. (Note that
    this routine does not clear the A-string before appending
    characters to it.)%
    %---------------%
    PROCEDURE(astrng);
```

```
   LOCAL char;
   REF astrng;
   af();
   disarm();
   LOOP
      BEGIN
      dn(&astrng);
      CASE char ← lookc() OF
         =CD:
            BEGIN
            input();
            GOTO STATE;
            END;
         =CA:
            BEGIN
            IF astrng.L = empty THEN incont(&astrng);
            RETURN;
            END;
         =C.:
            RETURN;
         =BC:
            IF astrng.L = empty THEN
               BEGIN
               input();
               BUMP [m];
               RETURN;
               END
            ELSE <INPFBK, bkc>(&astrng);
         =BW:
            IF astrng.L = empty THEN
               BEGIN
               input();
               BUMP [m];
               RETURN;
               END
            ELSE <INPFBK, bkw>(&astrng);
         ENDCASE *astrng* ← *astrng*, char;
      input();
      END;
   END.

(incont) PROCEDURE(string);
   %puts bugged text entity in string passed it%
   LOCAL TEXT POINTER bug1, bug2, ptr1, ptr2;
   REF string;
   INPUT BUG bug1 BUG bug2;
   <TXTEDT, tdr> ($bug1, $bug2, $ptr1, $ptr2);
   *string* ← ptr1 ptr2;
   dn(&string);
   RETURN;
   END.


(invsbl) PROCEDURE(astrng);
   CASE innmwd(astrng, FALSE, $vdr) OF
      =0:
```

```
        RETURN;
      =1:
        BEGIN
        BUMP [m];
        RETURN;
        END;
      ENDCASE
        BEGIN
        BUMP [m],[m];
        RETURN;
        END;
      END.

(inword) PROCEDURE(astrng);
   CASE innmwd(astrng, TRUE, $nmdr) OF
      =0:
        RETURN;
      =1:
        BEGIN
        BUMP [m];
        RETURN;
        END;
      ENDCASE
        BEGIN
        BUMP [m],[m];
        RETURN;
        END;
      END.

(inname) PROCEDURE(astrng);
   CASE innmwd(astrng, FALSE, $nmdr) OF
      =0:
        RETURN;
      =1:
        BEGIN
        BUMP [m];
        RETURN;
        END;
      ENDCASE
        BEGIN
        BUMP [m],[m];
        RETURN;
        END;
      END.

(innmwd)
   %This routine reads characters from the work station, appends them
   to a register, and displays them in the nameregister.  Alphabetic
   characters are forced to upper case before insertion into the
   A-string if the wordflag is false.  The argument should be the
   address of the register into which the A-string is to be put.
   (Note that this routine does not clear the A-string before
   appending characters to it.).%
   %---------------%
   PROCEDURE(astrng, wordflag, delimproc);
   LOCAL char;
```

```
   REF astrng, delimproc;
   af();
   disarm();
   LOOP
      BEGIN
      dn(&astrng);
      CASE char ← lookc() OF
         =CD:
            BEGIN
            input();
            GOTO STATE;
            END;
         =CA, =C.:
            BEGIN
            IF astrng.L = empty THEN
               BEGIN
               input();
               strbug($bl);
               delimproc($bl, $pl, $p2);
               IF wordflag THEN *astrng* ← pl p2
               ELSE *astrng* ← +pl p2;
               dn(&astrng);
               END;
            RETURN(0);
            END;
         =BC:
            IF astrng.L = empty THEN
               BEGIN
               input();
               RETURN(1);
               END
            ELSE <INPFBK, bkc>(&astrng);
         =BW:
            IF astrng.L = empty THEN
               BEGIN
               input();
               RETURN(1);
               END
            ELSE <INPFBK, bkw>(&astrng);
         ENDCASE
            BEGIN
            IF NOT wordflag AND char IN ['a,'z] THEN
               char ← char - 40B;
            *astrng* ← *astrng*, char;
            END;
      input();
      END;
   END.

(inch) PROCEDURE(char);
   LOCAL nchar;
   af();
   disarm();
   CASE nchar ← lookc() OF
      =char:
         BEGIN
```

```
            input();
            RETURN;
            END;
         =CD:
            BEGIN
            input();
            GOTO STATE;
            END;
         =BC:
            BEGIN
            input();
            BUMP [m];
            RETURN ;
            END;
         ENDCASE
            BEGIN
            BUMP [m],[m];
            RETURN;
            END;
   END.

(danswer) %this procedure accepts a yes or no answer from the
keyboard, and returns with a 0 if the answer was negative, and a 1 if
it was positive%
   %-------------------%
   PROCEDURE;
   CASE inpcuc() OF
      = 'Y, = CA:
         BEGIN
         DSP(?OFF Yes);
         RETURN(TRUE);
         END;
      = 'N, = SP:
         BEGIN
         DSP(?OFF No);
         RETURN(FALSE)
         END;
      = CD: GOTO STATE;
      ENDCASE
         BEGIN
         qm();
         REPEAT;
         END;
   END.

(invspc)
   %This routine reads characters from tne work station and a
   backspace character, the routine returns false; if he enters a
   command accept, it returns true; command delete goes to state.  It
   sets the viewspecs large upon entry, and small, upon exit.%
   %---------------%
   PROCEDURE(srcdpa);
   LOCAL tgtdpa;
   savvspc(srcdpa);
   ltl();
   af();
```

```
   cdset(FALSE);
   CASE input() OF
      =CD:
         BEGIN
         savvspc(srcdpa);
         lts();
         an();
         GOTO STATE;
         END;
      =CA, =C.:
         BEGIN
         tgtdpa ← dsparea(lcda());
         lts();
         an();
         RETURN(tgtdpa);
         END;
      =BC:
         BEGIN
         an();
         arm();
         savvspc(srcdpa);
         lts();
         RETURN(FALSE);
         END;
      ENDCASE
         BEGIN
         sysvspec ← setlt(curchr, sysvspec, sysvspec[1] :
         sysvspec[1]);
         dspvsp(sysvspec, sysvspec[1], 3);
         REPEAT;
         END;
   END.

(inlevadj) PROCEDURE(astrng);
   %this procedure gets characters for LEVADJ.  It collects
   characters ontil it encounters something other than a U or D (or
   backspace).  It puts these characters into the string passed it
   and displays them in the name register.  Note that the string will
   be empty upon RETURN if the user does no level adjust.%
   %--------------%
   LOCAL char;
   REF astrng;
   *astrng* ← NULL;
   LOOP
      BEGIN
      dn(&astrng);
      CASE (char ← lookc()) OF
         =SP:
            BEGIN
            input();
            EXIT LOOP;
            END;
         ='u, ='d, IN ['0,'9]:
            *astrng* ← *astrng*, char;
         =BC:
            IF astrng.L > empty THEN BUMP DOWN astrng.L
```

```
            ELSE
               BEGIN
               input();
               BUMP [m];
               EXIT LOOP;
               END;
         =BW, =$ascbst: *astrng* ← NULL;
         =CD:
            BEGIN
            input();
            GOTO STATE;
            END;
         ENDCASE EXIT LOOP;
      input();
      END;
   RETURN;
   END.

%input BUG's CA%

   (in1ca) PROC(bg1);
      LOCAL cdot;
      REF bg1;
      INPUT BUG bg1 (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in2ca) PROC(bg1, bg2);
      LOCAL cdot;
      REF bg1, bg2;
      INPUT BUG bg1 BUG bg2 (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in3ca) PROC(bg1, bg2, bg3);
      LOCAL cdot;
      REF bg1, bg2, bg3;
      INPUT BUG bg1 BUG bg2 BUG bg3 (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in4ca) PROC(bg1, bg2, bg3, bg4);
      LOCAL cdot;
      REF bg1, bg2, bg3, bg4;
      INPUT BUG bg1 BUG bg2 BUG bg3 BUG bg4
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

%input BUG's TEXT CA%

   (in0tca) PROC(astrng);
      LOCAL cdot;
      REF astrng;
      INPUT TEXT astrng (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in1tca) PROC(bg1, astrng);
      LOCAL cdot;
      REF bg1, astrng;
```

```
      INPUT BUG bg1 TEXT astrng (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in2tca) PROC(bg1, bg2, astrng);
      LOCAL cdot;
      REF bg1, bg2, astrng;
      INPUT BUG bg1 BUG bg2 TEXT astrng (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in3tca) PROC(bg1, bg2, bg3, astrng);
      LOCAL cdot;
      REF bg1, bg2, bg3, astrng;
      INPUT BUG bg1 BUG bg2 BUG bg3 TEXT astrng
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

%input STID's CA%

   (in1sca) PROC(bg1);
      LOCAL cdot;
      REF bg1;
      INPUT STID bg1 (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in2sca) PROC(bg1, bg2);
      LOCAL cdot;
      REF bg1, bg2;
      INPUT STID bg1 STID bg2 (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in3sca) PROC(bg1, bg2, bg3);
      LOCAL cdot;
      REF bg1, bg2, bg3;
      INPUT STID bg1 STID bg2 STID bg3
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in4sca) PROC(bg1, bg2, bg3, bg4);
      LOCAL cdot;
      REF bg1, bg2, bg3, bg4;
      INPUT STID bg1 STID bg2 STID bg3 STID bg4
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

%input STID's TEXT CA%

   (in0stca) PROC(astrng);
      LOCAL cdot;
      REF astrng;
      INPUT TEXT astrng (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in1stca) PROC(bg1, astrng);
      LOCAL cdot;
      REF bg1, astrng;
      INPUT STID bg1 TEXT astrng (CA cdot ← 0; / C. cdot ← 1);
```

```
      RETURN (cdot) END.

   (in2stca) PROC(bg1, bg2, astrng);
      LOCAL cdot;
      REF bg1, bg2, astrng;
      INPUT STID bg1 STID bg2 TEXT astrng
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in3stca) PROC(bg1, bg2, bg3, astrng);
      LOCAL cdot;
      REF bg1, bg2, bg3, astrng;
      INPUT STID bg1 STID bg2 STID bg3 TEXT astrng
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

%input STID's LEVADJ%

   (in1sadj) PROC(bg1, astrng);
      LOCAL cdot;
      REF bg1, astrng;
      *astrng* ← NULL;
      INPUT STID bg1 LEVADJ astrng
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in2sadj) PROC(bg1, bg2, astrng);
      LOCAL cdot;
      REF bg1, bg2, astrng;
      *astrng* ← NULL;
      INPUT STID bg1 STID bg2 LEVADJ astrng
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

   (in3sadj) PROC(bg1, bg2, bg3, astrng);
      LOCAL cdot;
      REF bg1, bg2, bg3, astrng;
      *astrng* ← NULL;
      INPUT STID bg1 STID bg2 STID bg3 LEVADJ astrng
         (CA cdot ← 0; / C. cdot ← 1);
      RETURN (cdot) END.

%viewspec acceptor%

   (savvspc) PROCEDURE(dpa);
      %Given the address of a display area, this routine will save the
      viewspecs associated with the area in sysvspec and sysvspec[1]
      (for massaging by setlt).%
      %----------------%
      REF dpa;
      sysvspec ← dpa.davspec;
      sysvspec[1] ← dpa.davspc2;
      RETURN;
      END.

   (putvspc) PROCEDURE(dpa);
```

```
    %Given the address of a display area, this routine will put the
    viewspecs saved in sysvspec and sysvspec[1] in the display area.%
    %---------------%
    REF dpa;
    dpa.davspec ← sysvspec;
    dpa.davspc2 ← sysvspec[1];
    RETURN;
    END.

(setlt)
  %This routine adjusts the viewspecs in accord with characters entered
  during view specification.%
  %---------------%
  PROCEDURE(setchr, vs1, vs2);
  LOCAL settmp, vspec[2];
  vspec ← vs1;
  vspec[1] ← vs2;
  CASE setchr OF
    =CD:
      BEGIN
      IF vspec.vsrlev THEN GOTO rstlev;
      RETURN(vspec, vspec[1]);
      END;
    ='a: % l←l-1 %
      IF vspec.vsrlev THEN
        BEGIN
        IF vspec.vslev = 0 THEN
          BEGIN
          BUMP vspec.vslev;
          vspec.vslevd ← TRUE;
          END
        ELSE BUMP DOWN vspec.vslev;
        END
      ELSE IF vspec.vslev > 0 THEN BUMP DOWN vspec.vslev;
    ='b:  % l←l+1 %
      IF vspec.vslev <= 63 OR vspec.vslevd THEN
        BEGIN
        IF vspec.vslevd THEN
          BEGIN
          BUMP DOWN vspec.vslev;
          IF vspec.vslev = 0 THEN vspec.vslevd ← FALSE;
          END
        ELSE BUMP vspec.vslev;
        END;
    ='c:  % l←all %
      BEGIN
      vspec.vslev ← 63;
      vspec.vsrlev ← FALSE;
      END;
    ='d:  % l←1 %
      BEGIN
      vspec.vslev ← 1;
      vspec.vsrlev ← FALSE;
      END;
    ='e:  % l=rel %
      IF NOT vspec.vsrlev THEN
```

```
        BEGIN
        vspec.vsrlev ← vspec.vslev;
        vspec.vslev ← FALSE;
        vspec.vslevd ← FALSE;
        END
      ELSE
        BEGIN
        (rstlev):
        vspec.vslev ← vspec.vsrlev;
        vspec.vsrlev ← FALSE;
        vspec.vslevd ← FALSE;
        END;
    ='g:  % branch only on %
      BEGIN
      vspec.vsbrof ← TRUE;
      vspec.vsplxf ← FALSE;
      END;
    ='h:  % branch only / plex only off %
      BEGIN
      vspec.vsbrof ← FALSE;
      vspec.vsplxf ← FALSE;
      END;
    ='i:  % content analyzer success %
      BEGIN
      vspec.vscapf ← TRUE;
      vspec.vscakf ← FALSE;
      END;
    ='j:  % content analyzer off %
      BEGIN
      vspec.vscapf ← FALSE;
      vspec.vscakf ← FALSE;
      END;
    ='k:  % content analyzer k flag %
      % only use content analyzer for first statement in sequence %
      BEGIN
      vspec.vscakf ← TRUE;
      vspec.vscapf ← FALSE;
      END;
    ='l:  % plex only on %
      BEGIN
      vspec.vsplxf ← TRUE;
      vspec.vsbrof ← FALSE;
      END;
    ='m:  vspec.vsstnf ← TRUE;  % location numbers on %
    ='n:  vspec.vsstnf ← FALSE;  % location numbers off %
    ='o:  vspec.vsfrzf ← TRUE;  % frozen on %
    ='p:  vspec.vsfrzf ← FALSE;  % frozen off %
    ='q:  % t←t-1 %
      IF vspec.vstrnc > 0 THEN BUMP DOWN vspec.vstrnc;
    ='r:  % t←t+1 %
      IF vspec.vstrnc < 63 THEN BUMP vspec.vstrnc;
    ='s:  % t←all %
      vspec.vstrnc ← 63;
    ='t:  % t←1 %
      vspec.vstrnc ← 1;
    ='u:  % display area formatter on %
```

```
      vspec.vsdaft ← TRUE;
    ='v:  % display area formatter off %
      vspec.vsdaft ← FALSE;
    ='w:  % l=t=all %
      BEGIN
      vspec.vstrnc ← 63;
      vspec.vslev ← 63;
      vspec.vsrlev ← FALSE;
      END;
    ='x:  % l=t=l %
      BEGIN
      vspec.vstrnc ← vspec.vslev ← 1;
      vspec.vsrlev ← FALSE;
      END;
    ='y:  vspec.vsblkf ← TRUE;  % blank line on %
    ='z:  vspec.vsblkf ← FALSE;  % blank line off %
    ='A:  vspec.vsindf ← TRUE;  % indenting on %
    ='B:  vspec.vsindf ← FALSE;  % indenting off %
    ='C:  vspec.vsnamf ← TRUE;  % names on %
    ='D:  vspec.vsnamf ← FALSE;  % names off %
    ='G:  vspec.vsstnr ← TRUE;  % statement numbers on right %
    ='H:  vspec.vsstnr ← FALSE;  % statement numbers on left %
    ='K:  vspec.vsidtf ← TRUE;  % initials, date, on %
    ='L:  vspec.vsidtf ← FALSE;  % initials, date, off %
    ='M:  vspec.vsmkrf ← TRUE;  % markers on %
    ='N:  vspec.vsmkrf ← FALSE;  % markers off %
    ='O:  vspec.vsusqf ← TRUE;  % user sequence generator on %
    ='P:  vspec.vsusqf ← FALSE;  % user sequence generator off %
    ENDCASE NULL;
  RETURN(vspec, vspec[1]);
  END.

(softcd)
  %This routine is generally used by the bug accepting routines. It is
  called when they encounter a backspace character while waiting for a
  bug mark.  This routine goes to state if there are no lower entries
  on the stack; otherwise, it pops the return stack, uses DELBM to
  erase the most recent bug mark, and returns.%
  %---------------%
  PROCEDURE;
  delbm();
  RETURN;
  END.

(specreg)
  %Spec a register.  This procedure converts a string or a statement
  identifier to a t-pointer.  The conversion algorithm depends on the
  first character in the register, and on the parameter passed as the
  second argument,  If a string is being matched, the routine expects
  as a third argument the stid at which the search should begin.
    If the first character is a number, and a name is being speced,
    the register is assumed to contain a statement number, and FECHUX
    is used to convert the A-string to a STID.  Otherwise the register
    is assumed to contain either a word or a string to be matched.
    If TYPE (the second argument) is
    1, the register is assumed to contain a name;
```

```
      2, the register is assumed to contain a word;
      3 or 6, the register is assumed to contain a string;
      4, the register is assumed to contain a statement identifier;
      In all of these cases LOOKUP is called to find the STID.%
   %---------------%
   PROCEDURE(astrng, type, ptr);
   REF astrng, ptr;
   IF astrng.L = empty THEN
      BEGIN
      ptr.stpsid ← origin;
      ptr[1] ← 1;
      END
   ELSE IF *astrng*[frstchr] IN ['0, '9] AND type = name THEN
      BEGIN %number%
      ptr ← <NLS, SEQGEN, fechux>(&astrng, ptr.stfile);
      ptr[1] ← 1;
      END
   ELSE <NLS, JUMP, lookup>(&ptr, &astrng, type);
   RETURN;
   END.

(strbug) PROCEDURE(ptr);
   %this routine will svae the current bug mark in the pointer passed
   it.%
   %-------------%
   REF ptr;
   IF bugreg = endfil THEN
      ptr ← pbug(bugreg[1] : ptr[1])
   ELSE
      BEGIN
      ptr ← bugreg;
      ptr[1] ← bugreg[1];
      END;
   RETURN;
   END.

FINISH of PRMSPC L10
```

RECFIL

```
<NLS>RECFIL.NLS;103, 28-DEC-71 12:24 JDH ;
FILE recfil  % L10 to <REL-NLS>RECFIL %
DECLARE rectfg = 1, pc = 1, detflg = 0;
REF tda;
SET gjinf = 13B, deldf = 67B, gpjfn = 206B, spjfn = 207B;
(checkfile) PROC (astr, z2);
   %open the file named in astr, an do anything necessary to assure that
   it is left as an unlocked and legal NLS file%
   %Return false if there was no trace of file, otherwise true%
   %-------------------%
   LOCAL fileno, retry, dirno, type, stid, jfn;
   LOCAL TEXT POINTER z1, z3;
   LOCAL STRING lookst[50];
   REF astr, z2;
   dismes(1, &astr);
   retry ← -1;
   (chloop):
   BUMP retry;
   stid ← 0;
   stid.stpsid ← origin;
   IF NOT stid.stfile ← fileno ← rawopen(&astr, TRUE) THEN RETURN(0);
   ON SIGNAL
       =-5, =-6: %Bad File or I/O data error%
         IF (retry = 0) AND [flntadr(fileno)].flpart THEN
            BEGIN %Unlock it, and see if the original is any good%
            unlkfile(fileno);
            dismes(1, $"File Unlocked");
               %do you want to leave the message up??%
            GOTO chloop;
            END
         ELSE
            IF sysgnl = -5 THEN
               BEGIN
               IF NOT [flntadr(fileno)].flexis THEN
                  IF NOT fileno ← rawopen(&astr, TRUE) THEN RETURN(0);
               cer(fileno);
               updtfl(fileno, 1);
               setaccess(fileno, jrnlaccess);
               close(fileno);
               dismes(2, $"File Reset");
               RETURN(0);
               END
            ELSE
               BEGIN %try to go back a version on an I/O data error%
               %Type out a message%
                  dismes(1, $"File Deleted (I/O Data Error)--backed up
                  to previous version");
               %Delete the file%
                  IF NOT writeout(fileno, $lit)
                  OR NOT sysclose((jfn ← [flntadr(fileno)].florig := 0)
                  .V 4B11) THEN RETURN(0);
                  delfil(fileno);
                  r1 ← jfn;
                  IF NOT SKIP !JSYS delf THEN RETURN(0);
               %Now call checkfile again to check the previous version%
                  RETURN(checkfile(&astr, &z2));
```

```
            END;
      ELSE;
   cef(fileno, 1); %verify file%
   updtfl(fileno, 1);
   WHILE NOT (FIND z2 $NP ENDCHR) DO %check for content or name%
      BEGIN
      type ← 0;
      IF FIND z2 "Name: " $NP ↑z1 L $LD ↑z2 ↑z3 THEN
         type ← name
      ELSE
         IF FIND z2 "Content: " $NP ↑z1 [';] ↑z2 ↑z3 ←z3 THEN
            type ← contnt
         ELSE
            FIND z2  $NP [NP/ENDCHR] ↑z2;
      IF type THEN
         BEGIN
         *lookst* ← z1 z3;
         FIND SF(stid) ↑z1;
         lookup($z1, $lookst, type);
         IF z1 = endfil THEN
            BEGIN
            dismes(1, $"Logical File Error");
            RETURN(0);
            END;
         END;
      END;
   setaccess(fileno, jrnlaccess); %Reset access in case it has changed
   mysteriously%
   close(fileno);
   RETURN(TRUE);
   END.

(xrecovf) PROC;
   %recover NLS files at start up time%
   LOCAL fileno, stid, faterr;
   LOCAL TEXT POINTER z1, z2;
   faterr ← 0;
   ON SIGNAL
      =-5: %We Got here on a badfile, and who knows how%
         IF fileno = bfilno THEN
            BEGIN
            close(fileno := 0);
            typeas($"File List Bad File");
            r1 ← 4B5; !JSYS haltf;
            END
         ELSE
            BEGIN
            lockjo(1);
            typeas($"Bad File--");
            typeas($lit);
            r1 ← 4B5; !JSYS haltf;
            END;
      ELSE
         BEGIN
         crlf();
         typeas(sysmsg);
```

```
        crlf();
        typeas($"Fatal Error");
        r1 ← 4B5;
        !JSYS haltf;
        END;
lockjo(0); %Lock Journal%
%Detatch and Set up logging TTY 0 as primary output file%
    IF autostrt THEN
        BEGIN
        crlf();
        typeas($"Detaching Slinker");
        !JSYS dtach;
        r1 ← 4B5;
        !JSYS gpjfn;
        r2.rh ← 4B5; %tty 0%
        r1 ← 4B5;
        !JSYS spjfn;
        END;
r1 ← getsbn($"RECOVF");
!JSYS setnm;
conjdir(TRUE); %Connect to JOunal Directory%
enablaccess(0, jrnlaccess);
IF enablw() = -1 THEN
    BEGIN
    crlf();
    typeas($"You need WHEEL capability to run recover files");
    r1 ← 4B5;
    !JSYS haltf;
    END;
IF (fileno ← rawopen(jflname($"recoverlist"), FALSE)) = 0 THEN
    BEGIN
    typeas($"File List open fail");
    r1 ← 4B5;
    !JSYS haltf;
    END;
stid ← 0;
stid.stfile ← fileno;
stid.stpsid ← origin;
stid ← getsub(stid);
LOOP
    BEGIN
    IF stid.stpsid = origin THEN EXIT;
    CCPOS SF(stid);
    FIND $NP ↑z1 (/NP/ < CH > / SE(stid)) ↑z2;
    *lit* ← z1 z2;
    IF NOT checkfile(jflname($lit), $z2) THEN BUMP faterr;
    stid ← getsuc(stid);
    END;
close(fileno:=0);
%Unlock Journal and proceed if no fatal errors%
    IF faterr THEN
        BEGIN
        typeas($"Fatal File Errors--Journal Left Locked");
        lockjo(1); %just to be sure%
        !JSYS haltf;
        END;
```

```
      unlkjo(-1); %allow Journal usage%
   END.

(recovf)PROCEDURE(libf);
   % bits of libf as follows%
      % 1B: run recover files %
      % 2B: run on-line distribution %
      % 4B: run slinker %
      % 10B: dettach, go to sleep, on hour run on-line dist. and slinker
      %
   IF (libf .A 1B) THEN xrecovf();
   IF NOT (libf .A 16B) THEN RETURN;
   IF autostrt
      THEN BEGIN
         %Set controlling tty up as primary output again%
         r1 ← 4B5;
         !JSYS gpjfn;
         r2.rh ← -1;
         r1 ← 4B5;
         !JSYS spjfn;
         END
      ELSE IF (libf .A 10B) THEN BEGIN
         crlf();
         typeas($"Detaching");
         !JSYS dtach;
         END;
   slinker(libf);
   r1 ← 4B5;
   !JSYS haltf;
   END.

(slinker) PROC (libf);
   %Slink around and update journal files, do online distribution, and
   garbage collect distribution file%
   %--------------------%
   LOCAL stidl, stid, curtime, verfuf;
   LOCAL TEXT POINTER z1, z2;
   stid ← stidl ← 0;
   ON SIGNAL
      =-5: %We Got here on a badfile, and who knows how%
         IF stid.stfile = bfilno THEN
            BEGIN
            close(stid.stfile  := 0);
            snkerr($"File List Bad File");
            END
         ELSE
            IF stidl.stfile =  bfilno THEN
               (lockfiles): BEGIN
               lockjo(1);
               *lit* ← "Bad File--Probably ", *jnamstr*, EOL, *lit*;
               snkerr($lit);
               END;
      =-6: %I/O data error%
         BEGIN
         *lit* ← *[sysmsg]*;
         GOTO lockfiles;
```

```
      END;
   ELSE
      snkerr(sysmsg);
LOOP
   BEGIN
   IF (libf .A 2B) THEN oldist(); %Distribute on-line documents%
   IF NOT (libf .A 14B) THEN RETURN;
   IF NOT (libf .A 4B) THEN  GOTO sinkslp;
   r1 ← getsbn($"SLINKR");
   !JSYS setnm;
   enablaccess(0, jrnlaccess);
   IF (stid ← openlock( 0,jflname($"Jfiles"))) = 0 THEN
      snkerr($"File List open fail");
   verfuf←2;
   LOOP
      BEGIN
      IF stid.stpsid = origin THEN BEGIN
         IF verfuf
            THEN BEGIN
               stid←getsub(stid);
               BUMP DOWN verfuf;
            END ELSE EXIT;
         END;
      CCPOS SF(stid);
      FIND $NP ↑z1 (/NP/ / SE(stid)) ↑z2;
      *lit* ← z1 z2;
      stid1 ← openlock(0,jflname($lit));
      crng(TRUE, stid1.stfile);
      csdb(TRUE, stid1.stfile);
      ckstrc(stid1.stfile);
      IF verfuf THEN BEGIN
         updtfl(stid1.stfile, 1);
         setaccess(stid1.stfile, jrnlaccess); %reset access%
         END;
      close(stid1.stfile);
      stid ← getsuc(stid);
      END;
   close(stid.stfile);
   %Now expunge if flag set%
      IF flagut(5, $tstfg) THEN
         BEGIN
         !JSYS gjinf;
         r1 ← r2; %connected directory number%
         !JSYS deldf; %expunge files%
         END;
   unlkjo(1); %alright to use it now%
   (sinkslp):
   IF NOT (libf .A 10B) THEN RETURN;
   libf←17B;
   r1 ← getsbn($"SNKSLP");
   !JSYS setnm; %set up name for sleepinng%
   r2 ← -1;
   r4 ← 0;
   !JSYS odcnv;
   IF r4 = -1 THEN r1 ← 1800000 %system does not have time%
   ELSE
```

```
      BEGIN
      CASE (curtime ← r4.rh) OF
         > 21 * 3600: r1 ← 24 * 3600 - curtime + 5 * 3600;
         < 5 * 3600: r1 ← 5 * 3600 - curtime;
         ENDCASE
            IF (r1 ← 3600 - curtime MOD 3600) < 300 %5 minutes% THEN
               r1 ← r1 + 3600;
                  %Do not run it this time if less than 5 minutes
                  until time%
      r1 ← r1 * 1000; %Convert to ms%
      END;
   !JSYS disms;
   END;
END.

(snkerr)PROC(astr);
   %Accepts an error string in astr, blaps it to tty0, tty30, and
   primary output, an halts%
   REF astr;
   %First set up subsystem name to snkerr%
      r1 ← getsbn($"SNKERR");
      !JSYS setnm;
   %Now blap error to tty 0 and 30%
      specttyout(0, &astr);
      specttyout(30B, &astr);
   %now type it to primary output%
      crlf();
      typeas(&astr);
   %Now die%
      !JSYS haltf;
   END.

(nlsutilty) PROC(modeflg);
   %Do utilty type of thiings for NLS%
   %----------------%
   %Mode flg = 4 for run detatched, 5 for run attatched with separate
   primary output file, and 6 for run attatched with tty pimary output%
   LOCAL stid1, stid, stiddone, action, rtime, doany, fl, pmyjfn;
   LOCAL TEXT POINTER z1, z2, z3, z4;
   LOCAL STRING hourst[5], minstr[5], filnms[50];
   REF fl;
   stid ← stid1 ← 0;
   ON SIGNAL
      =-5: %We Got here on a badfile, and who knows how%
         IF stid.stfile = bfilno THEN
            BEGIN
            close(stid.stfile  := 0);
            typeas($"Task List File Bad");
            r1 ← 4B5; !JSYS haltf;
            END
         ELSE
            err($"Bad File");
      ELSE
         BEGIN
         typeas(sysmsg);
         typeas($"Fatal Error");
```

```
      r1 ← 4B5; !JSYS haltf;
      END;
%connect to NLS directory%
   r1 ← 1;
   r2 ← chbptr(0) + $"NLS";
   !JSYS stdir;
      GOTO conerr;
      GOTO conerr;
   r1.lh ← 0;
   r2 ← chbptr(0) + $"NLS";
   IF NOT SKIP !JSYS cndir THEN
      BEGIN
      (conerr):
      crlf();
      typeas($"Directory Conect Fail");
      !JSYS haltf;
      END;
%Set up Primmary Output File%
   IF modeflg # 6 THEN
      pmyjfn ← setupop(0, modeflg);
LOOP
   BEGIN
   r1 ← getsbn($"UTILTY");
   !JSYS setnm;
   stid ← opntasks();
   CCPOS SF(stid);
   FIND ↑z1;
   *auxlit* ← "TODO";
   lookup($z1, $auxlit, name);
   IF z1 = endfil THEN
      BEGIN
      typeas($"No To Do Branch in Tasks File");
      r1 ← 4B5;
      !JSYS haltf;
      END;
   stid1 ← getsub(stid ← z1);
   *auxlit* ← "DONE";
   lookup($z1, $auxlit, name);
   IF z1 = endfil THEN
      stiddone ← cis(stid, $"(DONE) Tasks Which Are Done", sucdir)
   ELSE stiddone ← z1;
   STATE ← utstate, spec;
   ON SIGNAL ELSE
      BEGIN
      IF stid.stfile = 0 THEN z1.stfile ← z2.stfile ← stid1.stfile ←
      stid.stfile ← (opntasks()).stfile;% re-Open tasks file and
      update file number%
      *datesr* ← NULL;
      <NLS, AUXCOD, getdat>($datesr);
      ST stid1 ← z1 z2, SP, *datesr*, EOL, *[sysmsg]*;
      cms(stiddone, stid1 := getsuc(stid1), down);
      %Type out message if not detatched%
         IF modeflg # 4 THEN
            BEGIN
            *xlit* ← EOL, "*****", *[sysmsg]*, 0;
            r1 ← 18M;
```

```
            r2 ← chbptr(0) + $xlit;
            r3 ← 0;
            !JSYS sout;
            END;
      GOTO STATE;
      END;
   LOOP
      BEGIN
      IF stid1 = stid THEN EXIT;
      %Now parse command statement%
         CCPOS SF(stid1);
         %First see if it is being done by a parallel process%
            IF FIND ["In Progress"] THEN GOTO notnow;
         %First, find what we are supposed to do%
            action ← IF FIND ["Compile"] THEN 1
               ELSE IF FIND ["Print"] THEN 2
               ELSE 0;
            IF action = 0 THEN err($"Illegal Command");
         %Now get file name, and time%
            IF NOT FIND $NP ↑z1 1$PT ↑z2 THEN
               err($"Illegal Command Statement");
            *filnms* ← z1 z2;
            IF FIND $NP ↑z1 1$2 D ↑z2 ': ↑z3  1$2 D ↑z4 THEN
               BEGIN %Time to do it%
               *hourst* ← z1 z2; %Hours since midnight%
               *minstr* ← z3 z4; %Minutes%
               rtime ← VALUE($hourst)*3600 + VALUE($minstr) * 60;
               END
            ELSE rtime ← 0;
               %Do it now if nothing specified%
         %Set z1 and z2 to command%
            FIND SF(stid1) ↑z1 SE(stid1) ↑z2;
      %Now execute it if time%
         r2 ← -1;
         r4 ← 0;
         !JSYS odcnv;
         IF r4.rh >= rtime THEN
            BEGIN
            CASE action OF
               =1: %compile%
                  BEGIN
                  r2 ← -1;
                  r4 ← 0;
                  !JSYS odcnv;
                  IF (r4.rh NOT IN [12*3600, 13*3600]) AND
                  (r4.rh < 20*3600) AND
                  (r4.rh MOD 3600) > 600 THEN
                     %GOTO notnow; % NULL;
                        %Since it is not 1200 to 1300, after 20:00.
                        or within 10 min past hour do not do
                        compilations now%
                  IF modeflg # 4 THEN
                     BEGIN %Type out message to controlling tty%
                     *xlit* ← 15B, 12B, "Compiling ", *filnms*, 0;
                     r1 ← 18M; %controlling tty%
                     r2 ← chbptr(0) + $xlit;
```

```
                    r3 ← 0;
                    !JSYS sout;
                    END;
                %Now mark and release tasks file so another
                parallel process can use it%
                    ST stid1 ← z1 z2, EOL, "In Progress";
                    close(stid.stfile := 0);
                runcmp($filnms);
                END;
             =2: outptr($filnms);
                %process print requests independent of current
                time%
             ENDCASE err($"Illegal Command");
          %Now fix up command statement and move it to done list%
             IF NOT stid.stfile THEN z1.stfile ← z2.stfile ←
             stid1.stfile ← stid.stfile ← (opntasks()).stfile;
             %re-open tasks file and update file numbers%
             *datesr* ← NULL;
             getdat($datesr);
                %date/time of complettion%
             ST stid1 ← z1 z2, EOL, "Completed at ", *datesr*;
             cms(stiddone, stid1:= getsuc(stid1), down);
          END
       ELSE
          (notnow): stid1 ← getsuc(stid1);
    END;
closeu(stid.stfile := 0);
%Now close any files which got left open by error%
    %We Don't want to lose primary output stuff, so restore to tty,
    and re-open immediately after close all files%
       IF modeflg # 6 THEN
          BEGIN
          r1 ← 4B5;
          !JSYS gpjfn;
          r2.rh ← r2.lh;
          r1 ← 4B5;
          !JSYS spjfn;
          END;
    %Now expunge if flag set%
       IF flagut(5, $tstfg) THEN
          BEGIN
          !JSYS gjinf;
          r1 ← r2; %connected directory number%
          !JSYS deldf; %expunge files%
          END;
    %Now kill self if not running detatched%
       IF %modeflg # 4% TRUE  THEN %Always die for now%
          BEGIN
          crlf();
          typeas($"Done");
          !JSYS haltf;
          END;
    r1 ← -1;
    IF SKIP !JSYS closf THEN
       BEGIN
       IF modeflg # 6 THEN
```

```
            pmyjfn ← setupop(0, modeflg); %set utilty Primary Output%
         %re-open initial file%
            %original%
               &fl ← flntadr(tda.dacsp.stfile);
               *lit* ← *(fl.flastr)*;
               IF (fl.florig ← sgtjfn(getgtjflg(read, origff,
               dfltvrs), $lit, $lit2)) = 0
               OR NOT sysopen(fl.florig, read, random, $lit) THEN
                  BEGIN
                  typeas($"Could not re-open original");
                  rl ← 4B5;
                  !JSYS haltf;
                  END;
            %Partial Copy%
               IF fl.flpart THEN
                  BEGIN
                  *lit* ← *(fl.flpcst)*;
                  IF (fl.flpart ← sgtjfn(getgtjflg(write, origff,
                  dfltvrs), $lit, $lit2)) = 0
                  OR NOT sysopen(fl.flpart, readwrite, random, $lit)
                  THEN
                     BEGIN
                     typeas($"Could not re-open original");
                     rl ← 4B5;
                     !JSYS haltf;
                     END;
                  END;
            END
         ELSE setupop(pmyjfn);
      rl ← getsbn($"UTISLP");
      !JSYS setnm;
      !JSYS gtad;
      CASE rl.rh OF
         IN (8*3600, 12*3600):  rl ← 12*3600-rl.rh; %until 1200%
         IN (12*3600, 13*3600):  rl ← 60; % 1 minute%
         IN (13*3600, 20*3600):  rl ← 20*3600-rl.rh; %until 2000%
         IN (20*3600, 8*3600):  rl ← 600; % 10 minutes%
         ENDCASE err($"Irby error");
      rl ← rl*1000; %convert to milliseconds%
      !JSYS disms;
   END;
   END.

(opntasks)PROC;
   %open the tasks file for nlsutilty%
   %Return stid of origin, halt on error%
   LOCAL stid;
   IF (stid ← openlock(0, $"<NLS>TASKS.NLS")) = 0 THEN
      BEGIN
      typeas($"Task List open fail");
      rl ← 4B5;
      !JSYS haltf;
      END;
   RETURN(stid);
   END.
(setupop) PROC (jfn,modeflg);
```

```
   %This procedure opens a file called UTILTY-OUTPUT" IF jfn = 0, an
   sets it up as the primary outpu file.%
   %It returns the JFNN of the new Primmary outout file%
   LOCAL STRING filnms[50];
   %open a sequential file for primary output%
      IF jfn = 0 THEN
         BEGIN
         *filnms* ← "U-OUTPUT.TXT", EOL;
         IF NOT (jfn ← sgtjfn(getgtjflgs(write, 0, dfltvrs), $filnms,
         $lit))
         OR NOT sysopen(jfn, write, chrtyp, $lit)
         OR NOT sysclose(jfn .V 4B11)
         OR NOT sysopen(jfn, write, chrtyp, $lit) THEN err($lit);
         %Type out file version number%
            jfnstr(jfn, $lit);
            *lit* ← "Primary Output File is ", *lit*;
            crlf();
            typeas($lit);
         r1 ← jfn;
         r2 ← 3;
         IF NOT SKIP !JSYS delnf THEN NULL;
         END;
   %Type out detatching message%
      IF modeflg = 4 THEN
         BEGIN
         crlf();
         typeas($"Detatching Utilty");
         !JSYS dtach;
         END;
   %Now make it the primary output file%
      r1 ← 4B5;
      !JSYS gpjfn; %get primary JFN's%
      r2.rh ← jfn;
      r1 ← 4B5;
      !JSYS spjfn; %set Primary JFN's%
   RETURN(jfn);
   END.
(runcmp) PROC (fnmstr);
   LOCAL stid, errcount;
   LOCAL TEXT POINTER z1, z2, z3, z4;
   LOCAL STRING cmpnam[50], rfilnm[50], lookst[5]; %For rel-file name%
   %This procedur accepts the name of a source code file, and compiles
   it with the compiler indicated be first satemen in e file, to te
   indicated rel-file.  If anything in the firs statement does not match
   the accepted syntax ( ['percent] $NP COMPILER [percent] < $NP $LD >
   FILE percent) THen if the first statement is te FILE statement of an
   L10 Program, the file is compiled as L10 to the relfile under REL-NLS
   with the same name as te FILE.
   %
   REF fnmstr;
   stid ← orgstid;
   ON SIGNAL
      =ofilerr: err($"Rel-File open fail");
      =prcerr: err($"Compiler open fail or no such compiler");
      ELSE
         BEGIN %error%
```

```
      IF stid.stfile THEN close(stid.stfile  := 0);
      END;
   *fnmstr* ← *fnmstr*, ".NLS",  EOL;
   IF NOT (stid.stfile ← rawopen(&fnmstr, 0)) THEN err($"Cannot Open
   Source File");; %Open Source code file%
   makeptr(stid, $z1);
   *lookst* ← "FILE";
   lookup($z1, $lookst, contnt);
   IF  z1 = endfil THEN err($"Bad Source Code File");
   stid ← z1;
   %Get r1-file name and compiler%
      CCPOS SF(stid);
      IF NOT FIND ['%] $NP ↑z1 $LD ↑z2 ['%] < CH $NP ↑z4 [NP] > CH ↑z3
      THEN
         IF NOT FIND  $NP "FILE" $NP ↑z1 $LD ↑z2 THEN
            err($"Illegal Format Source File")
         ELSE
            BEGIN
            *rfilnm* ← "<REL-NLS>", z1 z2, ".REL", EOL;
            *cmpnam* ←  "L10";
            END
      ELSE
         BEGIN
         *rfilnm* ←  z3 z4, EOL;
         *cmpnam* ←  z1 z2;
         END;
   %call compiler%
      tda.dacsp ← stid;
      IF (errcount ← coc($cmpnam, $rfilnm, &tda)) > 0 THEN
         BEGIN
         *lit* ← STRING(errcount), " Errors";
         err($lit);
         END;
   %Close source file%
      close(stid.stfile := 0);
   %Return normally%
   RETURN;
   END.


(outptr) PROC (fnmstr);
   LOCAL stid, pfjfn;
   LOCAL TEXT POINTER z1, z2;
   LOCAL STRING  pfilnm[50], filnmstring[50]; %For rel-file name%
   % This procedure accepts address of astring contaaning a file name
   and does an output quickprint on the file, and then copies the result
   to the line printer. %
   REF fnmstr;
   stid ← orgstid;
   ON SIGNAL ELSE
      BEGIN
      IF stid.stfile THEN close (stid.stfile  := 0);
      IF lptjfn THEN sysclose (lptjfn  := 0);
      END;
   *filnmstring* ← *fnmstr*, ".NLS",  EOL;
   IF NOT (stid.stfile ← rawopen ($filnmstring, 0)) THEN
```

```
      err ($"Cannot Open File"); %Open Source code file%
   %Get output file name%
      CCPOS SF(*filnmstring*);
      FIND $NP  ↑z1 SE(z1) ↑z2 >  z1 ([']/ ↑z1/)  ['.] ↑z2 ←z2;
      *pfilnm* ← z1 z2, ".TXT", EOL;
   %call Output Quickprint%
      tda.dacsp ← stid;
      coq ($pfilnm, &tda);
   %Close Source File%
      close(stid.stfile := 0);
   %Now copy to lpt%
      pfjfn ← sgtjfn (getgtjflag (write, 0, oldvrsn), $pfilnm, $lit);
      IF NOT sysopen (pfjfn, readwrite, chrtyp, $lit) THEN err($lit);
      lptjfn ← 0;
      *ptstr* ← "LPT:";
      opnlpt();
      copylpt (0, pfjfn, 0, -1);
      sysclose (lptjfn := 0);
   %now close and delete file%
      sysclose (pfjfn .V 4B11); %Because TENEX doesn't work like book
      says%
      rl ← pfjfn;
      IF NOT SKIP !JSYS delf THEN NULL;
   %Return normally%
   RETURN;
   END.


FINISH
(Catalog) Catalog of NLS Procedures Called
   (autojo,closeu) (recfil,SETUPOP)  (ioexec,sgtjfn) (ioexec,getgtjflg)
   (ioexec,sysopen) (recfil,SETUPOP)  (ioexec,lgtjfn) (ioexec,gprjfn)
   (auxcod,processor) (ioexec,sysclose) (auxcod,opinit)
   (dspgen,dismes) (ioexec,rawopen) (ioexec,flntadr) (ioexec,unlkfile)
   (coren1,crf) (ioexec,updtfl) (ioexec,setaccess) (ioexec,close)
   (coren1,cef) (inpfbk,typeas) (autojo,lockjo) (autojo,conjdir)
   (ioexec,enablaccess) (inpfbk,crlf) (filmnp,getsub) (recfil,CHECKFILE)
    (autojo,jflname) (filmnp,getsuc) (autojo,unlkjo) (recfil,SLINKER)
   (auxcod,getsbn) (ioexec,openlock) (verify,crng) (verify,csdb)
   (verify,ckstrc) (coren1,crerr) (recfil,SETUPOP)  (jump,lookup)
   (coren1,cis) (coren1,cms) (recfil,RUNCMP)  (recfil,OUTPTR)
   (jnldel,enablw) (jnldel,oldist) (jnldel,opnlpt) (jnldel,copylpt)
(fcat) Formatted Catalog Branch
   (copylpt)  (jnldel,copylpt)
   (opnlpt)  (jnldel,opnlpt)
   (oldist)  (jnldel,oldist)
   (enablw)  (jnldel,enablw)
   (OUTPTR)  (recfil,OUTPTR)
   (RUNCMP)  (recfil,RUNCMP)
   (cms)  (coren1,cms)
   (cis)  (coren1,cis)
   (lookup)  (jump,lookup)
   (SETUPOP)  (recfil,SETUPOP)
   (crerr)  (coren1,crerr)
   (ckstrc)  (verify,ckstrc)
```

```
(csdb)  (verify,csdb)
(crng)  (verify,crng)
(openlock)  (ioexec,openlock)
(getsbn)  (auxcod,getsbn)
(SLINKER)  (recfil,SLINKER)
(unlkjo)  (autojo,unlkjo)
(getsuc)  (filmnp,getsuc)
(jflname)  (autojo,jflname)
(CHECKFILE)  (recfil,CHECKFILE)
(getsub)  (filmnp,getsub)
(crlf)  (inpfbk,crlf)
(enablaccess)  (ioexec,enablaccess)
(conjdir)  (autojo,conjdir)
(lockjo)  (autojo,lockjo)
(typeas)  (inpfbk,typeas)
(cef)  (corenl,cef)
(close)  (ioexec,close)
(setaccess)  (ioexec,setaccess)
(updtfl)  (ioexec,updtfl)
(crf)  (corenl,crf)
(unlkfile)  (ioexec,unlkfile)
(flntadr)  (ioexec,flntadr)
(rawopen)  (ioexec,rawopen)
(dismes)  (dspgen,dismes)
(opinit)  (auxcod,opinit)
(sysclose)  (ioexec,sysclose)
(processor)  (auxcod,processor)
(gprjfn)  (ioexec,gprjfn)
(lgtjfn)  (ioexec,lgtjfn)
(SETUPOP)  (recfil,SETUPOP)
(sysopen)  (ioexec,sysopen)
(getgtjflg)  (ioexec,getgtjflg)
(sgtjfn)  (ioexec,sgtjfn)
(SETUPOP)  (recfil,SETUPOP)
(closeu)  (autojo,closeu)
```