(astr)	<nine, psedt2, 01321>	LOCAL	5P2B1A
(bkbp)	<nine, psedt2, 01986>	PROCEDURE	5S3C
(ccurcon)	<nine, psedt2, 0436>	LOCAL	5F2A
(ccurioc)	<nine, psedt2, 0411>	LOCAL	5E2A
(compactfile)	<nine, psedt2, 03035>	PROCEDURE	5X2D1
(conbp)	<nine, psedt2, 02246>	PROCEDURE	5S3C
(copfil)	<nine, psedt2, 03131>	PROCEDURE	5X2E1
(coprng)	<nine, psedt2, 03230>	PROCEDURE	5X2E2
(cprocess)	<nine, psedt2, 04992>	PROCEDURE	5G2A
(cprofil)	<nine, psedt2, 03926>	PROCEDURE	5O2G
(creiallsta)	<nine, psedt2, 0562>	LOCAL	5I2B
(creista)	<nine, psedt2, 0535>	LOCAL	5I2A
(crensidfil)	<nine, psedt2, 0615>	LOCAL	5J2A
(crepgro)	<nine, psedt2, 0819>	LOCAL	5L2B
(crepsta)	<nine, psedt2, 0805>	LOCAL	5L2A
(creptex)	<nine, psedt2, 0849>	LOCAL	5L2D
(crestemmod)	<nine, psedt2, 0968>	LOCAL	5M2A
(crstty)	<nine, psedt2, 04612>	PROCEDURE	5M2B
(csetextname)	<nine, psedt2, 01180>	LOCAL	5O2E
(csetlindef)	<nine, psedt2, 01135>	LOCAL	5O2A
(csetngro)	<nine, psedt2, 01152>	LOCAL	5O2B
(csetnsta)	<nine, psedt2, 01162>	LOCAL	5O2C
(csettemmod)	<nine, psedt2, 01170>	LOCAL	5O2D
(csetty)	<nine, psedt2, 04624>	PROCEDURE	5O2F
(cshodir)	<nine, psedt2, 04081>	PROCEDURE	5P2H
(cshodskspa)	<nine, psedt2, 01319>	LOCAL	5P2B
(cshofilsta)	<nine, psedt2, 01383>	LOCAL	5P2C
(cshofrring)	<nine, psedt2, 04801>	LOCAL	5P2D
(cshomarfil)	<nine, psedt2, 01456>	LOCAL	5P2F
(cshonsta)	<nine, psedt2, 01462>	PROCEDURE	5P2G
(cshosrring)	<nine, psedt2, 01424>	PROCEDURE	5P2E
(cshoviespe)	<nine, psedt2, 01311>	PROCEDURE	5P2A
(csubgro)	<nine, psedt2, 01842>	LOCAL	5S2A
(csubsta)	<nine, psedt2, 01863>	LOCAL	5S2B
(ctcfm)	<nine, psedt2, 03712>	LOCAL	4L
(ctcmk)	<nine, psedt2, 03711>	LOCAL	4K
(ctcptz)	<nine, psedt2, 03709>	LOCAL	4I
(ctcsp)	<nine, psedt2, 03706>	LOCAL	4F
(cttrz)	<nine, psedt2, 03707>	LOCAL	4G
(ctxcfm)	<nine, psedt2, 03713>	LOCAL	4M
(ctmkr)	<nine, psedt2, 03710>	LOCAL	4J
(ctragro)	<nine, psedt2, 02378>	LOCAL	5U2A
(ctrasta)	<nine, psedt2, 02489>	LOCAL	5U2B
(ctratex)	<nine, psedt2, 02626>	PROCEDURE	5U2D
(ctridir)	<nine, psedt2, 04373>	PROCEDURE	5V2A
(cundfil)	<nine, psedt2, 04459>	PROCEDURE	5W2A
(cundmodfil)	<nine, psedt2, 04579>	PROCEDURE	5W2B
(cupdfil)	<nine, psedt2, 02707>	LOCAL	5X2A
(done)	<nine, psedt2, 01955>	LOCAL	5S2B19
(dsply)	<nine, psedt2, 03925>	EXT CONSTANT =51	4O
(endlin)	<nine, psedt2, 03708>	LOCAL	4H
(fltprc)	<nine, psedt2, 03612>	PROCEDURE	5S3A
(fndtab)	<nine, psedt2, 03279>	PROCEDURE	5A2D
(gndsjfn)	<nine, psedt2, 01361>	LOCAL	5P2B8A6J
(inpt)	<nine, psedt2, 03702>	REF	4B
(noct)	<nine, psedt2, 03705>	LOCAL	4E

(noenlf)	<nine, psedt2, 01199>	LOCAL	5O2E9H1
(nofile)	<nine, psedt2, 03704>	LOCAL	4D
(noteq)	<nine, psedt2, 01951>	LOCAL	5S2B17G4
(numvers)	<nine, psedt2, 04302>	LOCAL	5V1A1A1
(padstmt)	<nine, psedt2, 03384>	PROCEDURE	5B3A
(parselink)	<nine, psedt2, 01206>	CATCHPHRASE	5O2E15
(putchr)	<nine, psedt2, 0318>	LOCAL	5A2B
(rawchr)	<nine, psedt2, 03701>	REF	4A
(rensorg)	<nine, psedt2, 04928>	LOCAL	5J2B
(repeatloop)	<nine, psedt2, 04844>	CATCHPHRASE	5P2D14
(rjrepn)	<nine, psedt2, 0878>	PROCEDURE	5L3C
(rjshift)	<nine, psedt2, 0870>	PROCEDURE	5L3B
(rplist)	<nine, psedt2, 0835>	LOCAL	5L2C
(rptloop)	<nine, psedt2, 04841>	LOCAL	5P2D11S
(sbdelck)	<nine, psedt2, 01991>	PROCEDURE	5S3D
(sbinit)	<nine, psedt2, 02045>	PROCEDURE	5S3F
(sbpush)	<nine, psedt2, 02204>	PROCEDURE	5S3M
(setrot)	<nine, psedt2, 03586>	PROCEDURE	5X2D2
(skipni)	<nine, psedt2, 01954>	LOCAL	5S2B18
(stbpget)	<nine, psedt2, 01976>	PROCEDURE	5S3B
(step)	<nine, psedt2, 01906>	LOCAL	5S2B17
(step1)	<nine, psedt2, 01904>	LOCAL	5S2B16
(sub1dsp)	<nine, psedt2, 02058>	PROCEDURE	5S3I
(sub2dsp)	<nine, psedt2, 02077>	PROCEDURE	5S3J
(subpsave)	<nine, psedt2, 02184>	PROCEDURE	5S3L
(subsinit)	<nine, psedt2, 02000>	PROC	5S3E
(tda)	<nine, psedt2, 03703>	REF	4C
(trnchk)	<nine, psedt2, 0859>	PROCEDURE	5L3A
(trntst)	<nine, psedt2, 02616>	LOCAL	5U2C
(typwrtr)	<nine, psedt2, 03924>	EXT CONSTANT =52	4N
(updtf1)	<nine, psedt2, 02721>	PROCEDURE	5X2B
(updtpg)	<nine, psedt2, 03003>	PROCEDURE	5X2C
(window)	<nine, psedt2, 03263>	LOCAL	5Y1A1A1
(xchksimtty)	<nine, psedt2, 03907>	PROCEDURE	5Q1B
(xdlpnt)	<nine, psedt2, 04051>	LOCAL	5P2I1
(xmousespecs)	<nine, psedt2, 04740>	PROCEDURE	5Z1A
(xprint)	<nine, psedt2, 016>	LOCAL	5A1A
(xprocess)	<nine, psedt2, 04649>	PROCEDURE	5G1A
(xprtcc)	<nine, psedt2, 0428>	PROCEDURE	5F1A
(xprtnext)	<nine, psedt2, 0366>	PROCEDURE	5C1A
(xprtprev)	<nine, psedt2, 0385>	PROCEDURE	5D1A
(xprtsnum)	<nine, psedt2, 0403>	PROCEDURE	5E1A
(xprtstmt)	<nine, psedt2, 0452>	PROCEDURE	5B1A
(xputtext)	<nine, psedt2, 0461>	LOCAL	5H1A
(xrelease)	<nine, psedt2, 0503>	LOCAL	5I1A
(xrenumber)	<nine, psedt2, 0585>	LOCAL	5J1A
(xrepeatsearch)	<nine, psedt2, 0629>	PROCEDURE	5K1A
(xreplace)	<nine, psedt2, 0658>	PROCEDURE	5L1A
(xreset)	<nine, psedt2, 0898>	LOCAL	5M1A
(xset)	<nine, psedt2, 01022>	PROCEDURE	5O1A
(xshow)	<nine, psedt2, 03739>	PROCEDURE	5P1A
(xsimulate)	<nine, psedt2, 03890>	PROCEDURE	5Q1A
(xsort)	<nine, psedt2, 01723>	LOCAL	5R1A
(xsubresolve)	<nine, psedt2, 03524>	PROCEDURE	5S1B
(xsubstitute)	<nine, psedt2, 01757>	PROCEDURE	5S1A
(xterm)	<nine, psedt2, 03506>	LOCAL	5T1

(xtranspose)	<nine, psedt2, 02256>	LOCAL	5U1A
(xtrim)	<nine, psedt2, 04297>	LOCAL	5V1A
(xundelete)	<nine, psedt2, 04325>	LOCAL	5W1A
(xupdate)	<nine, psedt2, 02655>	LOCAL	5X1A
(xverify)	<nine, psedt2, 03260>	LOCAL	5Y1A

```
< NINE, PSEDT2.NLS;40, >, 24-Jul-78 22:46 GAS2 ;;;;
FILE psedt2 % <ARCSUBSYS>L109 to <RELNINE>psedt2 %% (arcsubsys,L109,)
(RELNINE,psedt2.rel,) %
%Allow system interface calls.  They exist in the procedures gpget,
gpadjbsz, coutproc, xreset, xshow, csubsta, (1 each, I think) and
cshodskspa (7  occurrences)%
   ALLOW!
% Compile-time switches %
   SET NSW = FALSE;  %TRUE for NSW => (RELNINE,wmpsedt2.rel,)%
% Declarations %
   (rawchr) REF;                                                        4A
   (inpt) REF;                                                          4B
   (tda) REF;                                                           4C
   (nofile) = 0;                                                        4D
   (noct)   = 0;                                                        4E
   (ctcsp)  = 1;                                                        4F
   (cttrz)  = 2;                                                        4G
   (endlin) = -1;                                                       4H
   (ctcpfz) = 3;                                                        4I
   (ctmkr)  = 8;                                                        4J
   (ctcmk)  = 9;                                                        4K
   (ctcfm)  = 11;                                                       4L
   (ctlcfm) = 15;                                                       4M
   (typwrtr) EXTERNAL CONSTANT = 52;                                    4N
   (dsply) EXTERNAL CONSTANT = 51;                                      4O

% EDITUR SUBSYSTEM %
   %print%
      %print PCP interface routine%
         (xprint) %Execute TNLS Print Command%                          5A1A
            PROCEDURE (
               %FORMALS%
                  entity REF,           %source type%
                  sourceptr REF,        %source pointer%
                  vs REF,               %viewspec record%
                  window                %number of last window bugged%
                  );
               LOCAL retry, enttype, da REF, source REF;
               LOCAL
                  vssav1, vssav2, cspsav,       %Save VS and SP%
                  vswrd1, vswrd2,               %to hold viewspecs%
                  savcacode, savusqcod,         %Save other da stuff%
                  pjb REF, pjba, pjbb, pjbc,    %Journal stuff%
                  pjl, pjbrad, pjvspec, pjbflag, pjoflag, jstid,
                  jsstid,                       %Starting stid for
                  Print Journal%
                  sw REF;                       %Seq work areas (addr)%
               %----------------------%
               enttype _ ELEM #entity# [cwtype];
               IF &sourceptr THEN
                  BEGIN
                  &source _ ELEM #sourceptr#[tppair];
                  &da _ dsparea(ELEM #sourceptr#[wndw])
                  END
               ELSE &da _ dsparea(window);
               IF NOT &vs THEN
```

```
        BEGIN
        vswrd1 _ da.davspec;
        vswrd2 _ da.davspc2;
        END
      ELSE
        BEGIN
        vswrd1 _ vs;
        vswrd2 _ vs[1];
        END;
      CASE enttype OF
        = 29 %- statement -%:
          BEGIN
          curmkr _ source;  curmkr[1] _ source[1];
          da.davspec _ vswrd1;
          da.davspc2 _ vswrd1[1];
          cprint (&da, source, source, stmtv, cortn, 0);
          cspvs _ vswrd1;        %So  cmdfinish will set current
          values%
          cspvs[1] _ vswrd1[1];
          cspupdate _ &da;
          END;
        = 27 %- group -%, = 26 %- branch -%, = 28 %- plex -%:
          BEGIN
          curmkr _ source;  curmkr[1] _ source[1];
          da.davspec _ vswrd1;
          da.davspc2 _ vswrd1[1];
          cprint (&da, source, [&source+d2sel], groupv, cortn,
          0);
          cspvs _ vswrd1;        %So  cmdfinish will set current
          values%
          cspvs[1] _ vswrd1[1];
          cspupdate _ &da;
          END;
        = 52 %- rest -%, = 51 %- file -%:
          BEGIN
          cspsav _ da.dacsp;     %save current SP%
          vssav1 _ da.davspec;   %save current VS%
          vssav2 _ da.davspc2;
          da.davspec _ vswrd1.vs1;
          da.davspc2 _ vswrd1.vs2;
          IF enttype = 51 %+ file +% THEN da.dacsp.stpsid _
          orgstid;
          cprint (&da, da.dacsp, endfil, groupv, cortn, 0);
          da.dacsp _ cspsav;     %restore current SP%
          da.davspec _ vssav1;   %restore current VS%
          da.davspc2 _ vssav2;
          END;
        = 53 %- journal -%:
          BEGIN
          pjbflag _ pjoflag _ 0;
          %may delete these when pjb is made a calling
          parameter%
             &pjb _ $pjba;
             pjba_$"Journal"; % pjbb_$"Info"; pjbc_$"Action"; %
          pj!_1; %used to be 3 when "Info" and "Action" weren't
          commented out above%
```

```
            jsstid _ da.davspec;
            pjvspec _ da.davspec;
            pjvspec.vsusqf _ TRUE;
            WHILE (pjl_pjl-1) >= 0 DO IF (pjbrad _ pjb[pjl]) THEN
               BEGIN
               IF (jstid _ namelook (jsstid, pjbrad) )= endfil
               THEN REPEAT LOOP;
               pjbflag _ TRUE;
               IF getsub(jstid)= jstid THEN REPEAT LOOP;
               pjoflag _ TRUE;
               &sw _ openseq( jstid, jstid, pjvspec, da.davspc2,
               $pjseqg, 0);
               cprint (&da, jstid, jstid, groupv, cortn, &sw);
               closeseq(&sw);
               END;
            IF NOT pjbflag THEN ABORT(nojrnl, $"No Journal
            Branches");
            IF NOT pjoflag THEN ABORT(nomail, $"No Mail");
            END;
         ENDCASE
         IF NOT donthelp THEN
            BEGIN
            retry _ HELP (badarg, $"The type argument was
            incorrect.  The allowed values are Branch, File,
            Group, Journal, Plex, Rest, or Statement.", cindex);
            REPEAT CASE (retry);
            END
         ELSE ABORT (badarg, $"The type argument was
         incorrect.");
      RETURN;
      END.

%print core routines%

   (putchr) % put a character into an a-string;  return the
   column increment for normal characters, endlin for end of
   line, 0 for null characters%                                  5A2B
      PROCEDURE (da,char,column,astrng);
      LOCAL
         colinc,  %column increment%
         newcol,  %new column for tabs%
         cntrlstr, %address of non-printing char string%
         count;   %counter for processing chars in a-string%
      REF astrng, da;
      %-----------------%
      %
      char _ trnslo[char];
      %
      colinc _ 0;
      IF oshift AND char IN ['A, 'Z] THEN
         BEGIN
         BUMP colinc;
         %
         *astrng* _ *astrng*, trnslo[cshift];
         %
         char _ char + 40B;
```

```
        END;
      CASE char OF
        = nullch: RETURN(0);
        = TAB:
          BEGIN
          IF (newcol _ fndtab(&da, column%+1%)) > da.damcol
          THEN
            RETURN(endlin);
              %end of line%
          colinc _ newcol - column - 1;
            %number of spaces to emit%
          *astrng* _ *astrng*, TAB;
          RETURN(colinc);
          END;
        =CA, =LF, =CD, =BC, =BW, =$ascalt, IN [1B,32B], IN
        [34B,36B]: %an acceptable non-printing character%
          BEGIN
          cntrlstr _ npstrad(char);%get np representation%
          count _ 1;
          *astrng* _ *astrng*, *[cntrlstr]*;
          RETURN([cntrlstr].L);
          END;
        = CR, =EOL: RETURN(endlin); %end of line%
        ENDCASE %normal character%
          BEGIN
          *astrng* _ *astrng*, char;
          BUMP colinc;
          RETURN(colinc);
          END;
      END.


    (fndtab) PROCEDURE (da, column);  %find new tab position%  5A2D
      %This procedure returns the column position of the next
      tab, given a column position in column%
      %-----------------%
      LOCAL tabtbl[3];
      REF da;
      tabtbl _ da.datab0; %setup local tab table (adress shit
      $&#")%
      tabtbl[1] _ da.datab1;
      tabtbl[2] _ da.datab2;
      RETURN(nxtbit($tabtbl, column, 3));
      END.

%print current statement%
  %print current statement PCP interface routine%
    (xprtstmt) PROCEDURE( % Execute TNLS backslash command %   5B1A
      %FORMALS%
        window);   %number of window last bugged%
      LOCAL da REF;
      %-------------------%
      &da _ dsparea(window);
      IF da.daempty THEN err($"No file currently loaded.");
      crlf();
      cprint (&da, curmkr, curmkr, stmtv, regrtn, 0);
```

```
        RETURN;
        END.


    %print statement support routine%
       (padstmt) PROCEDURE (  %add spaces to the front of a statement
       to reflect its level, adjusting for viewspecs%                  583A
          %FORMALS%
             da REF,  %display area%
             stid,    %stid of statement being formatted%
             stmtstr REF);    %pointer to statement%
          LOCAL prindt, vspc1, vspc2;
          %--------------------%
          vspc1 _ da.davspec;
          vspc2 _ da.davspc2;
          prindt _          %indentation%
             CASE TRUE OF
                = vspc1.vsindf:
                   MAX (tpoffset, MIN (da.daind * (getlev(stid)-1) +
                   tpoffset, da.damind, spacestr.M));
                = vspc1.vsbrof, = vspc1.vsplxf:
                   MAX (tpoffset, 0);
                ENDCASE tpoffset;
             % TPOFFSET is a global which the user can set via
             Execute Viewchange.  This allows the user to control the
             left margin of his print out. %
          *stmtstr* _ *spacestr* [empty + 1 TO prindt], *stmtstr*;
          RETURN;
          END.

%print next statement%
    %print next statement PCP interface routine%
       (xprtnext) PROCEDURE( % Execute linefeed command %              5C1A
          % FORMAL ARGUMENTS %
             window); %number of window last bugged%
          LOCAL stid, da REF;
          LOCAL lvs[2];        % viewspec param record %
          LOCAL TEXT POINTER t1, t2;
          LOCAL STRING string[5];
          %----------%
          *string* _".n";
          FIND SF(*string*) ^t1 SE(*string*) ^t2;
          &da _ dsparea(window);
          IF da.daempty THEN err($"No file currently loaded.");
          caddexp($t1, $t2, &da, $curmkr);
          cprint (&da, curmkr, curmkr, stmtv, regrtn, 0);
          IF NOT nodisplay THEN dpset(dspjpf, curmkr, endfil,
          endfil);
          RETURN;
          END.

%print previous statement%
    %print previous statement PCP interface routine%
       (xprtprev) PROCEDURE( % Execute uparrow command %               5D1A
          % FORMAL ARGUMENTS %
             window); % number of last window bugged %
```

```
        LOCAL stid, da REF;
        LOCAL TEXT POINTER t1, t2;
        LOCAL STRING string[5];
        %----------%
        *string* _".b";
        FIND SF(*string*) ^t1 SE(*string*) ^t2;
        &da _ dsparea(window);
        IF da.daempty THEN err($"No file currently loaded.");
        caddexp($t1, $t2, &da, $curmkr);
        crlf();
        cprint (&da, curmkr, curmkr, stmtv, regrtn, 0);
        IF NOT nodisplay THEN dpset(dspjpf, curmkr, endfil,
        endfil);
        RETURN;
        END.

%print current location%
   %current location PCP interface routine%
      (xprtsnum) PROCEDURE( % Execute TNLS period command %      5E1A
        %FORMALS%
          rtnlist REF);    %return arg%
        LOCAL STRING astrng[50]; % collection string %
        %----------%
        IF tda.daempty THEN err($"No file currently loaded.");
        cspupdate _ FALSE;
        ccurloc($curmkr, $astrng );
        #rtnlist#[1] _ *astrng*;
        RETURN;
        END.

   %current location core routine%
      (ccurloc)              %Core NLS Current Location in File
      Command%                                                   5E2A
        PROCEDURE(tptr, string);
        REF tptr, string;
        IF tda.davspec.vssidf
          THEN % use SID's %
            *string* _ " = 0", STRING( getsid( tptr ) )
          ELSE % use statement number %
            BEGIN
            *string* _ " = ";
            fechno(tptr, &string);
            END;
        *string* _ *string*, " +", STRING(tptr[1]);
        RETURN;
        END.

%print current context%
   %current context PCP interface routine%
      (xprtcc) PROCEDURE( % Execute TNLS slash command %         5F1A
        %FORMALS%
          rtnlist REF);    %return arg%
        LOCAL STRING string[100];
        %--------------------%
        IF tda.daempty THEN err($"No file currently loaded.");
        cspupdate _ FALSE;
```

```
        ccurcon( $curmkr, $string );
        #rtnlist#[1] _ *string*;
        RETURN;
        END.

    %current context core routine%
        (ccurcon)              %Core NLS Current Context in File
        Command%                                                    5F2A
            PROCEDURE(tptr, string);
            LOCAL TEXT POINTER tcm, ptr;
            REF tptr, string;
            tcm _ ptr _ tptr;
            tcm[1] _ tptr[1];
            ptr[1] _ MAX(1, tcm[1]-tslshchars);
            *string* _ SP, ptr tcm, LF, "==>";
            FIND SE(tcm) ^ptr;
            ptr[1] _ MIN(tcm[1]+tslshchars+1, ptr[1]);
            *string* _ *string*, tcm ptr, "  ";
            RETURN;
            END.

%process commands%
    %process commands PCP interface routine%
        (xprocess)   % CL: ; Protocol Interface Routine for Process
        Commands %
        PROCEDURE (dest REF, rtnlist REF);                          5G1A
            % Procedure description
                FUNCTION
                    This xroutine prepares a sequential file from the
                    structure passed in dest.
                    From each NLS statement, an ACSII string is formed
                    and terminated by an end-of-statement character,
                    <^0>.
                ARGUMENTS
                    dest--REF-DSEL of the process commands structure
                    selected.
                    rtnlist--REF-pointer to result list.
                RESULTS
                    String containing sequential file name is returned to
                    the FE in rtnlist.
                NON-STANDARD CONTROL
                    none
                GLOBALS
                    none
                %
            % Declarations %
                LOCAL
                    destination REF,
                    curstid, endstid;  % stid range for cmds %
                LOCAL STRING
                    prcmdfile[100];
            % determine start and end stids %
                &destination _ ELEM #dest#[tppair];
                curstid _ grptst(destination, destination[d2sel] :
                endstid);
                endstid _ getend(endstid);
```

```
        % call core routine %
           cprocess(curstid, endstid, $prcmdfile);
        % prepare results for FE %
           #rtnlist#[1] _ *prcmdfile*;
        % Return %
           RETURN;
        END.

    %process commands core routine%
       (cprocess)   % CL: ; Protocol Interface Routine for Process
       Commands %
       PROCEDURE (curstid, endstid, prcmdfile REF);                    5G2A
          % Procedure description
             FUNCTION
                This xroutine prepares a sequential file from the
                sequence of statements grouped by curstid and
                endstid.  This  is not an NLS group, rather a
                sequential set of statements at any level.
                From each NLS statement, an ACSII string is formed
                and terminated by an end-of-statement character,
                <^D>.
             ARGUMENTS
                curstid--INT-starting stid of set of statements
                selected.
                endstid--INT-ending stid of set of statements
                selected.
                prcmdfile--REF-addr of string to get sequential file
                name or 0 if failure.
             RESULTS
                none
             NON-STANDARD CONTROL
                none
             GLOBALS
                none
             %
          % Declarations %
             LOCAL
                jfn;  % of prc cmds sequential file %
             LOCAL CONSTANT
                eosflg = 17B;
             LOCAL TEXT POINTER
                tp1, tp2;
          % open prc cmds file %
             *prcmdfile* _ "<, *userstr*, ">, "TEMPSPRC.CMDS";
             IF NOT jfn _ sgtjfn(6B11 .V 1B6, &prcmdfile, $lit)  THEN
                err($lit);
             IF NOT sysopen(jfn, append, chrtyp, $lit) THEN
                BEGIN
                reljfn(jfn);
                err($lit);
                END;
          % copy all statements of the structure, ignoring statement
          names %
             LOOP
                BEGIN
                FIND SE(curstid) ^tp2;
```

```
            tp1 _ tp2;
            tp1[1] _ fchtxt(curstid); % move past statement name
            %
            IF tp1[1] # tp2[1] THEN
               BEGIN % nonzero string length %
               *lit* _ tp1 tp2, eosflg;
               appseq(jfn, $lit);
               END;
            IF curstid = endstid THEN EXIT LOOP;
            curstid _ getnxt(curstid);
            END;
         % close prc cmds file %
            IF NOT sysclose(jfn .V 4B11, $lit) THEN
               BEGIN
               reljfn(jfn);
               err($lit);
               END;
         % prepare results for FE %
            filsize(jfn : bytes);
            IF bytes THEN
               BEGIN % return file name %
               jfnstr(jfn, &prcmdfile);
               END
            ELSE
               BEGIN % return NULL and delete file %
               dltfile(jfn);
               &prcmdfile _ 0;
               END;
         % release jfn %
            reljfn(jfn);
         % Return %
            RETURN;
         END.

%puttext%
   %puttext PCP interface routine%
      (xputtext) %Execute Puttext Command%                              5H1A
         PROCEDURE (
            %FORMALS%
               destptr REF,  %pointer to destination%
               newstmt REF   %pointer to new statement%
               );
            LOCAL TEXT POINTER stmt1, stmt2;
            LOCAL retry REF, destid, level = 0;
         %--------------------%
         &retry _ &destptr;
         %Get an stid for the destination%
         CASE ELEM #retry#[cwtype] OF
            % branch inoperable - note double percents
            = 7 &&- oldfilename -%%:
               BEGIN %%Load that file or create a new one%%
               ....
               destid _ XXXX;        %%STID of origin statement&&
               END;
            %
            = 29 %- statement -%:
```

```
            destid _ [ELEM #retry#[tppair]];      %First word of
            first text pointer%
         ENDCASE
         IF NOT donthelp THEN
            BEGIN
            retry _ HELP (badarg, $"The selection type was
            incorrect.  The only allowed value is Statement.",
            cindex);
            REPEAT CASE;
            END
         ELSE ABORT (badarg, $"The selection type was
         incorrect.");

      WHILE newstmt.L DO %until *newstmt* is empty%
         BEGIN
         FIND SF(*newstmt*) ^stmt1 SE(*newstmt*) ^stmt2;
         destid _ cinssta(destid, level, stmt1, stmt2);   %Insert
         the statement%
         newstmt _ coreturn();
         END;

      %Done.  Any cleanup happens here%
      RETURN;
      END.
%release%
   %release PCP interface routine%
      (xrelease) %Execute Release Command%                        5I1A
         PROCEDURE (
            %FORMALS%
               entity REF,          %entity type%
               destptr REF          %destination pointer%
               );
            LOCAL da REF, destination REF, retry;
         %----------------------%
         IF &destptr THEN &destination _ ELEM #destptr#[tppair];
         &da _ dsparea(ELEM #destptr#[wndw]); %display area address%
         CASE ELEM #entity#[cwtype] OF
            = 51 %- frozen -%:
               BEGIN
               crelsta (&da, destination);
               END;
            = 52 %- all -%: %frozen statements%
               crelallsta(&da);
            ENDCASE
            IF NOT donthelp THEN
               BEGIN
               retry _ HELP (badarg, $"The type argument was
               incorrect.  The allowed values are All or Frozen.",
               cindex);
               REPEAT CASE (retry);
               END
            ELSE ABORT (badarg, $"The type argument was
            incorrect.");
         IF NOT nodisplay THEN
         %if frozen viewspec, recreate everything, else nothing%
            IF da.davspec.vsfrzf THEN
```

```
            dpset(dspallf,destination,endfil,endfil)
         ELSE dpset(dspno,endfil,endfil,endfil);
      RETURN;
      END.

%release core routine%
   (crelsta)                                                    5I2A
      %This routine searches the chain of frozen statements.
      If the stid passed it is in the frozen list for the display
      area passed it, the routine removes it from the list,
      squeezes the frozen list, and adds the deleted element to
      the frozen element free list.%
      %------------------%
      PROCEDURE(dpa, stid);
      LOCAL frzprev, frzelm;
      REF dpa, frzprev, frzelm;
      IF &frzprev _ &frzelm _ dpa.dafrzl THEN LOOP
         BEGIN
         IF frzelm.fzstid = stid THEN
            BEGIN
            IF &frzelm = &frzprev THEN %first item in list%
               dpa.dafrzl _ frzelm.fznext
            ELSE frzprev.fznext _ frzelm.fznext;
            frzelm.fzexis _ FALSE;
            frzelm.fznext _ fzfree := &frzelm;
            EXIT;
            END;
         &frzprev _ &frzelm;
         IF frzelm.fznext THEN &frzelm _ frzelm.fznext
         ELSE EXIT;
         END;
      RETURN;
      END.
   (crelallsta)                                                 5I2B
      %Given the address of a display area, this routine will
      free all of the frozen elements associated with the area.%
      %------------------%
      PROCEDURE(dpa);
      LOCAL frzelm;
      REF dpa, frzelm;
      IF &frzelm _ dpa.dafrzl THEN
         BEGIN
         LOOP
            BEGIN
            frzelm.fzexis _ FALSE;
            IF NOT frzelm.fznext THEN EXIT
            ELSE &frzelm _ frzelm.fznext;
            END;
         frzelm.fznext := fzfree := dpa.dafrzl := 0;
         END;
      RETURN;
      END.

%renumber%
   %renumber PCP interface routine%
      (xrenumber) %Execute Renumber Command%                    5J1A
```

```
        PROCEDURE (
           %FORMALS%
              window
              );
           LOCAL retry, destination REF, fileno;
           LOCAL STRING filestr[200];
        %----------------------%
        %
        IF &destptr THEN &destination _ ELEM #destptr#[tppair];
        CASE ELEM #destptr#[cwtype] OF
           = 2 %%- character -%%:
              fileno _ destination.stfile;
           = 7 %%- oldfilename -%%:
              BEGIN
              lnbfls (&destination, 0, $filestr);
              IF NOT (fileno _ cloafil($filestr)) THEN
                 ABORT (filenotfound, $"File Not Found.");
              END;
           ENDCASE
           IF NOT donthelp THEN
              BEGIN
              retry _ HELP (badarg, $"The type argument was
              incorrect.  The allowed values are Character or
              Oldfilename.", cindex);
              REPEAT CASE (retry);
              END
           ELSE ABORT (badarg, $"The type argument was
           incorrect.");
        %
        crensidfil(dspfile(window));
        IF NOT nodisplay THEN dpset(dspyes,
        [dsparea(window)].dacsp, endfil, endfil);
        RETURN;
        END.

  %renumber core routine%
     (crensidfil)   %Core NLS Renumber SID's in File Command%   5J2A
        PROCEDURE (fileno);
        LOCAL stid, sidc;  REF sidc;
        IF fileno NOT IN [1,filcnt] THEN ABORT(filebad,$"Illegal
        file");
        &sidc _ ($sidcnt-$filhed) + filehead[fileno];
        sidc _ 0;
        stid _ orgstid;  stid.stfile _ fileno;
        rensorg(stid, &sidc);
        RETURN;
        END.

     (rensorg)                                                  5J2B
        %renumbers sid's in branch hanging from origin stid%
           %sidc is address of sid counter%
        PROCEDURE (stid, sidc);
        LOCAL curpty, sdb, infstid;
        REF sidc, sdb;
        DO
           BEGIN
```

```
        stosid(stid, sidc _ sidc + 1);
        IF NOT getorf(stid) THEN
           BEGIN
           curpty _ getsdb(stid); %first property%
           LOOP
              BEGIN
              IF curpty.stpsdb=0 THEN EXIT;
              lodent(curpty, sdbtyp : &sdb);
              curpty.stpsdb _ sdb.spsdb;
              CASE sdb.sptype OF
                 =dhtyp, =chtyp: IF (infstid _ sdb.sitpsid) THEN
                    BEGIN
                    infstid.stfile _ stid.stfile;
                    rensorg (infstid, &sidc);
                    END;
                 ENDCASE NULL;
              END;
           END;
     END UNTIL (stid _ getnxt(stid)) = endfil;
  RETURN;
  END.   %%
```

```
%repeat search%
   %repeat search PCP interface routine%
      (xrepeatsearch) PROCEDURE( %Execute repeat last search
      command%                                                    5K1A
         % FORMAL ARGUMENTS %
            window          %number of last window bugged%
            );
         LOCAL da REF;
         LOCAL TEXT POINTER t1, t2, csp;
         LOCAL STRING treps[200];
         %-----------------------%
         CASE srchtype OF
            = wordtyp:
               *treps* _ "=w";
            = wordls:
               *treps* _ "=ws";
            = contnt:
               *treps* _ "=c";
            = contls:
               *treps* _ "=cs";
            ENDCASE ABORT(repeatsearch, $"<tab> valid only to repeat
            a previous search");
         *treps* _ ".n", "", *conreg*, "", *treps*;
         dismes(2, $treps);
         &da _ cspupdate _ dsparea(window) ;
         csp _ da.dacsp;
         csp[1] _ da.daccnt;
         FIND SF(*treps*) ^t1 SE(*treps*) ^t2;
         caddexp($t1, $t2, &da, $csp);
         curmkr _ csp;
         curmkr[1] _ csp[1];
         cspvs _ da.davspec;
         cspvs[1] _ da.davspc2;
         IF NOT nodisplay THEN dpset(dspjpf, curmkr, endfil,
         endfil);
         RETURN;
         END.

%replace%
   %replace PCP interface routine%
      (xreplace) %Execute Replace Command%  PROCEDURE             5L1A
         (
            %FORMALS%
               destptr REF,  %destination pointer%
               sourceptr REF,        %source pointer%
               txtrtn,       %if TRUE, return text of statement%
               tprtn         %if TRUE, return text pointer(s)%
               );
            LOCAL retry REF, endstid, grplist REF, delt, source REF,
            destination REF, lnktp1 REF, lnktp2 REF;
            LOCAL TEXT POINTER tp1, tp2;
            LOCAL STRING temp[40], linkstr[500], badmsg[200];
            LOCAL adstr[40];
            LOCAL LIST tplist [4], txtlist[1];
         %----------------------%
```

```
        IF &sourceptr THEN &source _ ELEM #sourceptr#[tppair];
        IF &destptr THEN &destination _ ELEM #destptr#[tppair];
        #tplist# _ ;
        #txtlist# _ ;
        &grplist _ $txtlist;
        CASE ELEM #sourceptr#[cwtype] OF
           = 30 %- link -%:
              BEGIN
              &lnktp1 _ &source+d1sel;
              &lnktp2 _ &source+d2sel;
              IF lnktp1.stastr THEN
                 BEGIN
                 *linkstr* _ lnktp1 lnktp2;
                 lnktp1.RH _ lnktp2.RH _ $linkstr;
                 IF NOT FIND SF(linkstr) $(SP/TAB) ("(/"</"--")
                 THEN
                    *linkstr* _ "<, *linkstr*;
                 IF NOT FIND SE(linkstr) < $(SP/TAB) (")/">) THEN
                    *linkstr* _ *linkstr*, ">;
                 END;
              lnkprs (&lnktp1, $adstr);
              lnktp1[0] _ adstr[ls];
                 lnktp1[1] _ adstr[ls+1];
              lnktp2[0] _ adstr[le];
                 lnktp2[1] _ adstr[le+1];
              END;
           ENDCASE;
        CASE ELEM #destptr#[cwtype] OF
           = 30 %- link -%:
              BEGIN
              lnkprs (&destination, $adstr);
              destination _ adstr[ls];
                 destination[1] _ adstr[ls+1];
              [&destination+d2sel] _ adstr[le];
                 [&destination+d2sel+1] _ adstr[le+1];
              END;
           = 8 %- integer -%:
              BEGIN
              delt _ destination[d2sel+1] - destination[1] -
              source[d2sel+1] +source[1];
              CASE delt OF
                 < 0:
                    BEGIN
                    LOOP
                       IF (delt := delt+1) >= 0 OR
                       (NOT FIND destination < SP SP ^destination
                       _destination) THEN EXIT LOOP;
                    END;
                 > 0:
                    BEGIN
                    tp1 _ source;
                    tp1[1] _ source[1];
                    tp2 _ source[d2sel];
                    tp2[1] _ source[d2sel+1];
                    *temp* _ NULL;
                    UNTIL (delt _ delt-1) < 0 DO *temp* _ *temp*,
```

```
                    SP;
                    *temp* _ *temp*, tp1 tp2;
                    FIND SF(*temp*) ^tp1 SE(*temp*) ^tp2;
                    source _ tp1;
                    source[1] _ tp1[1];
                    source[d2sel] _ tp2;
                    source[d2sel+1] _ tp2[1];
                    END;
                ENDCASE;
            END;
        ENDCASE;
    CASE ELEM #destptr#[cwtype] OF
        %text/structure entities%
            = 2 %- character -%, = 3 %- word -%, = 4 %- visible
            -%, = 11 %- invisible -%, = 30 %- link -%, = 8 %-
            integer -%, = 1 %- text -%, = 29 %- statement -%:
                BEGIN
                IF NOT nortnrings THEN
                    clist(ctmkr, destination.stfile,
                    source.stfile);
                IF NOT nodisplay THEN
                    dpset(dsprfmt, destination, endfil, endfil);
                curmkr _ destination;  curmkr[1] _
                destination[1]+source[d2sel+1]-source[1]-1;
                creptex(&destination, &destination+d2sel, &source,
                &source+d2sel);
                IF NOT nortnrings THEN clupdt();
                IF txtrtn THEN #txtlist# _ curmkr;
                END;
            = 26 %- branch -%, = 28 %- plex -%, = 27 %- group -%:
                BEGIN
                IF NOT nortnrings THEN
                    clist(ctlcfm, destination.stfile,
                    source.stfile);
                IF NOT nodisplay THEN
                    dpset(dsprfst, destination, endfil,
                    dpstp(destination));
                curmkr _ crepgro(NOT source.stastr, &destination,
                &destination+d2sel, &source, &source+d2sel);
                IF NOT nortnrings THEN clupdt();
                curmkr[1] _ 1;
                destination[1] _ 1;
                IF txtrtn THEN
                    BEGIN
                    &grplist _ alloclist(20);
                    endstid _ getend(curmkr);
                    #grplist# _ destination;
                    DO
                        BEGIN
                        #grplist# !_ getnxt(destination);
                        IF grplist.L = grplist.M THEN
                            &grplist _ alloclist(grplist.M + 20);
                        END
                    UNTIL grplist[grplist.L] = endstid;
                    END;
                END;
```

```
            ENDCASE
            IF NOT donthelp THEN
               BEGIN
               *badmsg* _ "The selection type was incorrect.  The
               allowed values are Branch, Character, ";
               *badmsg* _ *badmsg*, "Group, Integer, Invisible,
               Link, Plex, Statement, Text, Visible, or Word.";
               &retry _ HELP (badarg, $badmsg, cindex);
               source _ ELEM #retry#[tppair];
               REPEAT CASE (ELEM #retry#[cwtype]);
               END
            ELSE ABORT (badarg, $"The selection type was
            incorrect.");
         %
         IF tprtn THEN
            BEGIN
            #tplist# _ destination, destination[1];
            IF NOT ((destination = curmkr) AND (destination[1] =
            curmkr[1])) THEN
               #tplist# !_ curmkr, curmkr[1];
            END;
         IF tprtn THEN
            IF txtrtn THEN RETURN(grplist, tplist)
            ELSE RETURN (empty, tplist)
         ELSE
            IF txtrtn THEN RETURN(grplist, empty)
            ELSE RETURN;
         %
         RETURN;
         END.

   %replace core routine%
      (crepsta) %***%                %Core NLS Replace Statement
      Command%                                                    5L2A
         %copies name delimiters also%
         PROCEDURE (bugdit, atbug, bybug1, bybug2);
         LOCAL stid;
         REF atbug, bybug1, bybug2;
           %msntx();%
         freplist(atbug);
         IF bugdit THEN
            copplist(bybug1, atbug)
         ELSE %literal input%
            ST atbug _ bybug1 bybug2;
           %msftx();%
         RETURN(atbug);
         END.

      (crepgro)              %Core NLS Replace Group Command%     5L2B
         PROCEDURE (bugdit, atbug1, atbug2, bybug1, bybug2);
         LOCAL head, tail;
         REF atbug1, atbug2, bybug1, bybug2;
           %msntx();%
         atbug1 _ grptst(atbug1, atbug2 :atbug2);
         IF bugdit THEN
            BEGIN
```

```
        bybug1 _ grptst(bybug1, bybug2 :bybug2);
        head _ copgrp(atbug2, levsuc, bybug1, bybug2, FALSE:
        tail);
        cdelgro(atbug1, atbug2, 0);
        END
      ELSE head _ tail _ rpllit(atbug1, atbug2, &bybug1,
      &bybug2);
        %msftx();%
      RETURN(head, tail);
      END.

    (rpllit)                                                    5L2C
      %This routine replaces the group defined by the first two
      arguments passed it with a new statement containing the
      text in the astring passed it.  It returns the value of the
      new stid.
        First it gets a new stid, and inserts it as the
        successor of GRP2.  It then deletes the group bounded by
        GRP1, GRP2, and puts the text in the astring passed it
        in the new SDB.%
      %-------------%
      PROCEDURE(grp1, grp2, t1, t2);
      LOCAL stid; %stid for new statement%
      REF t1, t2;
      IF grp1.stfile # grp2.stfile THEN ABORT(badgrp,$"Illegal
      group");
      stid _ newrng(grp1.stfile);
      inss(grp2, stid, stid);
      cdelgro(grp1, grp2, 0);
      ST stid _ t1 t2;
      RETURN(stid);
      END.

    (creptex)              %Core NLS Replace Text Command%      5L2D
      PROCEDURE (bug1, bug2, bug3, bug4);
      REF bug1, bug2, bug3, bug4;
        %msntx();%
      cldtxt(&bug1, &bug2);
      ST bug1 _ SF(bug1) bug1,
        $bug3 bug4,       % $ => don't move markers %
        bug2 SE(bug1);
        %msftx();%
      RETURN;
      END.

  %routines for replace and transpose number%

    (trnchk) PROCEDURE (ptr1,ptr2,ptr3,ptr4);                   5L3A
      %moves pointers around for transpose number%
      LOCAL delt;
      delt _ [ptr2+1] - [ptr1+1] - [ptr4+1] + [ptr3+1];
      CASE delt OF
        < 0: rjshift(ptr1,delt);
        > 0: rjshift(ptr3,-delt);
        ENDCASE;
      RETURN;
```

```
      END.

   (rjshift) PROCEDURE (ptr,delt);                                    5L3P
      REF ptr;
      FIND ptr <;
      LOOP
         IF (delt := delt+1) >= 0 OR
         (NOT FIND SP ^ptr) THEN RETURN;
      END.

   (rjrepn) PROCEDURE (ptr1, ptr2, astring);                          5L3C
      LOCAL delt;
      REF astring;
      delt _ [ptr2+1] - [ptr1+1] - astring.L;
      CASE delt OF
         < 0: rjshift(ptr1,delt);
         > 0: BEGIN
            *num* _ *astring*;
            *astring* _ NULL;
            UNTIL (delt _ delt-1) < 0 DO *astring* _ *astring*,
            SP;
            *astring* _ *astring*, *num*;
            END;
         ENDCASE;
      RETURN;
      END.

%reset%
   %reset PCP interface routine%
      (xreset) %Execute Reset Command%                                5M1A
         PROCEDURE (
            %FORMALS%
               entity REF,            %entity type%
               destptr REF    %destination pointer%
               );
         LOCAL retry1, retry2 REF, destination REF, da REF, save,
         condir, cdnum, rhostn;
         LOCAL STRING filstring[200], badmsg[200];
         LOCAL TEXT POINTER tp1, tp2;
         %---------------------%
         IF &destptr THEN &destination _ ELEM #destptr#[tppair];
         &da _ lda();
         IF NOT nodisplay THEN dpset(dspno, endfil, endfil, endfil);
         CASE ELEM #entity#[cwtype] OF
            = 52 %- case -%: %mode%
               xsmode _ upcase;
            = 53 %- content -%: %Content Analysis%
               BEGIN
               da.dacacode _ 0;
               da.davspec.vscapf _ FALSE;
               END;
            = 30 %- link -%: %default for file%
               BEGIN
               !gjinf();      %get number of connected directory%
               cdnum _ P2.RH;
               condir _ chbmty + $filstring;
```

```
        IF NOT SKIP !dirst(condir, cdnum) THEN ABORT(0,
        $"directory number not in use");
        filstring.L _ ascizsize($condir);  %fix string
        length%
        FIND SF(*filstring*) ^tp1 SE(*filstring*) ^tp2;
        csetlindef(da.dacsp.stfile, $tp1, $tp2);
        END;
     = 32 %- name -%: %delimiters in destentity%
        BEGIN
        CASE ELEM #destptr#[cwtype] OF
           = 29 %- statement -%:
              csetngro(destination, destination, dfnmdl,
              dfnmdr, &da);
           = 27 %- group -%, = 26 %- branch -%, = 28 %- plex
           -%:
              csetngro(destination, [&destination+d2sel],
              dfnmdl, dfnmdr, &da);
           ENDCASE
           IF NOT donthelp THEN
              BEGIN
              &retry2 _ HELP (badarg, $"The selection type
              was incorrect.  The allowed values are Branch,
              Group, Plex, or Statement.", cindex);
              destination _ ELEM #retry2#[tppair];
              REPEAT CASE (ELEM #retry2#[cwtype]);
              END
           ELSE ABORT (badarg, $"The selection type was
           incorrect.");
        END;
     = 56 %- temporary -%: %modifications to file%
        crestemmod(da.dacsp.stfile, FALSE);
     = 52 %- tty -%: %window%
        BEGIN
           crstty();
        END;
     = 57 %- viewspecs -%:
        BEGIN
        cspupdate _ &da;
        cspvs _ stdvsp;
        cspvs[1] _ stdvsp[1];
        curmkr _ da.dacsp;  curmkr[1] _ da.daccnt;
        IF NOT nodisplay THEN dpset(dspyes, da.dacsp, endfil,
        endfil);
        END;
     = 58: %- buffer -%
        gpbsz(upgbdf);
     ENDCASE
     IF NOT donthelp THEN
        BEGIN
        *badmsg* _ "The type argument was incorrect.  The
        allowed values are Buffer, Case, Content, ";
        *badmsg* _ *badmsg*, "Link, Name, Temporary, and
        Viewspecs.";
        retry1 _ HELP (badarg, $badmsg, cindex);
        REPEAT CASE (retry1);
        END
```

```
            ELSE ABORT (badarg, $"The type argument was
            incorrect.");
         RETURN;
         END.

   %reset core routines%
      (crestemmod)   %Core NLS Reset Temporary Modifications
      Command%                                                    5M2A
         PROCEDURE (fileno, lock);
         LOCAL fl;
         LOCAL STRING filestr[100];
         REF fl;
         &fl _ flntadr(fileno);
         IF lock THEN
            IF NOT lockfile(fileno) THEN RETURN
            ELSE NULL
         ELSE
            BEGIN
            IF NOT fl.flbrws THEN
               BEGIN
               dismes (2, $"This file not in temporary modifications
               mode.");
               RETURN;
               END;
            unlkfile(fileno, FALSE);
            END;
         fl.flbrws _ FALSE;
         RETURN;
         END.

      (crstty)       % Core NLS Reset tty window Command %
      PROCEDURE;                                                  5M2B
         LOCAL oldda REF;
         REF ttyda, dttyda;
         % do nothing if tty window is the default %
            IF &ttyda = &dttyda THEN RETURN(TRUE);
         % update the ttyda pointer %
            &oldda _ &ttyda := &dttyda;
         % update the old  da %
            oldda.daseq _ oldda.dasuppress _ FALSE;
            dafrmt( &oldda, 0 );
         RETURN(TRUE);
         END.

%retrieve%           %moved to PSEDIT-EXTRA%
%set%
   %set PCP interface routine%
      (xset) %Execute Set Command%
      PROCEDURE (                                                 5D1A
         %FORMALS%
            entity REF,      %entity type%
            param1 REF,
            param2 REF,
            param3 REF,
            destptr REF,     %destination pointer%
            rtnlist REF   % result list %
```

```
    );
% Declarations %
    LOCAL retry1, retry2 REF, destination REF, mask, prot,
    count, i, rhostn, csize, hinc, vinc, da REF, end1, save,
    tp2 REF, stid, adstr[40], paramlist REF, parea1[10],
    parea2[10], temp1 REF, temp2 REF;
    LOCAL STRING sizestring[10], filstr[200], badmsg[200],
    patternstring[2048];
    LOCAL TEXT POINTER ttextpointer;
    LOCAL LIST settenlist[10];
%----------------------%
IF &destptr THEN &destination _ ELEM #destptr#[tppair];
IF NOT nodisplay THEN dpset(dspno, endfil, endfil, endfil);
CASE ELEM #entity#[cwtype] OF
    = 53 %- content -%: %pattern to%
        BEGIN
        getpstring(&param2, $patternstring);
        FIND SF(*patternstring*) ^ttextpointer;
        IF ccompile(
            53 %content%,
            ttextpointer, % starting stid %
            0, % default compiler %
            0, % default output file (none) %
            -1, % no messages %
            0, % no viewspecs -- include all %
            dsparea(&param3) % da % )
        THEN err($"illegal pattern  -- not instituted");
        END;
    = 59 %- external -%:  % names link file address %
        BEGIN
        stid _ orgstid;
        stid.stfile _ [dsparea(&param1)].dacsp.stfile;
        IF NOT nodisplay THEN dpset(dsprfmt, stid, endfil,
        endfil);
        &temp1 _ ELEM #param2#[tppair];
        lnkprs( &temp1, $adstr);
        csetextname( stid.stfile, $adstr );
        END;
    = 30 %- link -%: %default for file%
        BEGIN
        &temp1 _ ELEM #param2#[tppair];
        csetlindef(dspfile(&param1), &temp1, &temp1+d2sel);
        END;
    = 32 %- name -%: %delimiters in destentity%
        BEGIN                                              501A6D1
        IF &param2 THEN
            BEGIN                                          501A6D2A
            &temp1 _ ELEM #param2#[tppair];
            CCPOS temp1;
            temp1 _ READC;
            IF temp1 = ENDCHR THEN temp1 _ 0;
            END;
        IF &param3 THEN
            BEGIN                                          501A6D3A
            &temp2 _ ELEM #param3#[tppair];
            CCPOS temp2;
```

```
            temp2 _ READC;
            IF temp2 = ENDCHR THEN temp2 _ 0;
            END;
         curmkr _ destination;  curmkr[1] _ 1;
         CASE ELEM #param1#[cwtype] OF
            = 29 %- statement -%:
               csetnsta(destination, temp1, temp2);
            = 27 %- group -%, = 26 %- branch -%, = 28 %- plex
            -%:
               csetngro(destination, [&destination+d2sel],
               temp1, temp2, dsparea(ELEM #destptr#[wndw]));
            ENDCASE
            IF NOT donthelp THEN
               BEGIN
               &retry2 _ HELP (badarg, $"The selection type
               was incorrect.  The allowed values are Branch,
               Group, Plex, and Statement.", cindex);
               destination _ ELEM #retry1#[tppair];
               REPEAT CASE (ELEM #retry2#[cwtype]);
               END
            ELSE ABORT (badarg, $"The selection type was
            incorrect.");
         END;
      = 51 %- NLS private -%: %file%
         chprvsts (lcfile(), $psprivate);
      = 54 %- NLS  public -%: %file%
         chprvsts (lcfile(), $pspublic);
      = 56 %- temporary -%: %modifications to file%
         csettemmod(lcfile());
      = 52 %- tty -%: %window%
         csetty( &param1 );
      = 55 %- tenex -%: %protection for a file%
         BEGIN
         &temp1 _ ELEM #param1#[tppair];
         rhostn _ lnbfls( &temp1, 0, $tilstr);
         % parse the input %
            % convert parameter list %
               FOR i _ 1 UP UNTIL > param2.L DO
                  cnvarg(&param2, $settenlist, i);
            CASE ELEM #[ELEM #settenlist#[1]]# [cwtype] OF
               = 89 %- allow -%:
                  BEGIN
                  prot _ 0;
                  IF NOT (count _ param2.L - 2) THEN
                     err($"Illegal protection specified");
                  i _ 1;
                  WHILE i <= count DO
                     prot _ prot .V
                       (CASE  ELEM #[ELEM #settenlist#[2 + (i
                       := i + 1)]]# [cwtype] OF
                          = 81 %- read -%: 40B;
                          = 82 %- write -%: 20B;
                          = 83 %- execute -%: 10B;
                          = 84 %- append -%: 04B;
                          = 85 %- list -%: 02B;
                          = 52 %- all -%: 77B;
```

```
                    = 92 %- set -%:
                       ELEM #[ELEM #settenlist#[2 +
                       (i:=j+1)]]# [tppair];
                    ENDCASE 0 );
            prot _ prot .A 77B;
            CASE ELEM #[ELEM #settenlist#[2]]# [cwtype]
            OF
               = 70 %- self -%:
                  BEGIN
                  prot _ prot * 10000B;
                  mask _ 770000B;
                  END;
               = 27 %- group -%:
                  BEGIN
                  prot _ prot * 100B;
                  mask _ 007700B;
                  END;
               = 54 %- public -%:
                  BEGIN
                  prot _ prot * 1B;
                  mask _ 000077B;
                  END;
               ENDCASE
                  err($"Illegal protection specified");
            END;
         = 71 %- forbid -%:
            BEGIN
            prot _ 0;
            IF NOT (count _ settenlist.L - 2) THEN
               err($"Illegal protection specified");
            i _ 1;
            WHILE i <= count DO
                prot _ prot .V
                  (CASE ELEM #[ELEM #settenlist#[2+(i :=
                  i+1)]]# [cwtype] OF
                     = 81 %- read -%: 40B;
                     = 82 %- write -%: 20B;
                     = 83 %- execute -%: 10B;
                     = 84 %- append -%: 04B;
                     = 85 %- list -%: 02B;
                     = 52 %- all -%: 77B;
                     = 92 %- set -%:
                        ELEM #[ELEM
                        #settenlist#[2+(i:=i+1)]]#
                        [tppair];
                     ENDCASE 0 );
            prot _ prot .A 77B .X 77B;
            CASE ELEM #[ELEM #settenlist#[2]]# [cwtype]
            OF
               = 70 %- self -%:
                  BEGIN
                  prot _ prot * 10000B;
                  mask _ 770000B;
                  END;
               = 27 %- group -%:
                  BEGIN
```

```
                        prot _ prot * 100B;
                        mask _ 007700B;
                        END;
                     = 54 %- public -%:
                        BEGIN
                        prot _ prot * 1B;
                        mask _ 000077B;
                        END;
                     ENDCASE
                        err($"Illegal protection specified");
                  END;
               = 55 %- reset -%:
                  BEGIN
                  prot _ 777752B;
                  mask _ 18M;
                  END;
               = 51 %- private -%:
                  BEGIN
                  mask _ 18M;
                  prot _ CASE ELEM #ELEM #settenlist#[2]]#
                  [cwtype] OF
                     = 70 %- self -%: 770000B;
                     = 27 %- group -%: 777700B;
                     = 54 %- public -%: 777777B;
                     ENDCASE 777752B;
                  END;
               = 92 %- set -%:
                  BEGIN
                  mask _ 18M;
                  getpstring(#settenlist#[2], $sizestring);
                  prot _ VALUE($sizestring,8);
                  END;
               ENDCASE
                  err($"Illegal Protection Specified");
            *lit* _ NULL;
            cprofil(rhostn, $filstr, mask, prot, $lit);
            IF lit.L THEN
               *lit* _ "The protection of the following files has
               been changed:", CR, LF, *lit*
            ELSE
               *lit* _ "No files' protection changed";
            #rtnlist#[1] _ USE rtnstring($lit);
            END;
         = 57 %- viewspecs -%:
            BEGIN
            cspupdate _ dsparea( cwndow );
            IF &param1 THEN
               BEGIN
               cspvs _ param1;
               cspvs[1] _ param1[1];
               END;
            IF NOT nodisplay THEN dpset(dspyes,
            [cspupdate].dacsp, endfil, endfil);
            END;
         = 58 %- buffer -%:
            BEGIN
```

```
            %move size string to local string and convert to
            integer%
            &temp1 _ ELEM #param2#[tppair];
            &temp2 _ &temp1 + d2sel;
            *sizestring* _ temp1 temp2;
            csize _ VALUE($sizestring);
            %set the buffer to the new size%
            IF NOT gpbsz(csize) THEN
               ABORT(badbufsize, $"Invalid size for programs
               buffer.");
            END;
         ENDCASE
         IF NOT dorthelp THEN
            BEGIN
            *badmsg* _ "The type argument was incorrect.  The
            allowed values are Buffer, Content, ";
            *badmsg* _ *badmsg*, "External, Link, Name, Private,
            Public, Temporary, or Viewspecs.";
            retry1 _ HELP (badarg, $badmsg, cindex);
            REPEAT CASE (retry1);
            END
         ELSE ABORT (badarg, $"The type argument was
         incorrect.");
      % free list storage %
         #settenlist# _ ;
      RETURN;
      END.

%set core routine%
   (csetlindef)   %Core NLS Link Default of file Command%     502A
      PROCEDURE(fileno, d1, d2);
      LOCAL funoadr, dirno, stid, aring;
      LOCAL STRING locstr[50];
      REF d1, d2, aring;
      % make sure we have a locked file %
         stid _ origin;
         stid.stfile _ fileno;
         lodent( stid, rngtyp : &aring);
         aring.rsub _ aring.rsub;
      %get dir number%
         *locstr* _ d1 d2;
         dirno _ transdir($locstr);
      funoadr _ filhdr(fileno) + ($funo - $filhed);
      [funoadr] _ dirno;
      RETURN;
      END.

   (csetngro)              %Core NLS Set Name Delimiter Group
   Command%                                                  502B
      PROCEDURE (stid1, stid2, dlleft, dlrght, da);
      REF da;
      %uses the SEQUENCE GENERATOR%
        %msntx();%
      nmdlgset(stid1, stid2, dlleft, dlrght, &da);
         %fix this stupid code -- merge routines%
        %msftx();%
```

```
    RETURN;
    END.

(csetnsta)              %Core NLS Set Name Delimiter Statement
Command%                                                        502C
    PROCEDURE (stid, dlleft, dlrght);
      %msntx();%
    nmdlsset(stid, dlleft, dlrght);
       %fix this stupid code -- merge routines%
      %msftx();%
    RETURN;
    END.

(csettemmod)   %Core NLS Set Temporary Modifications to file%
                                                                502D
    PROCEDURE (fileno);
    LOCAL fl;
    REF fl;
    &fl _ flntadr(fileno);
    IF fl.flpart THEN
       ABORT(nowmod,$"You are already modifying this file!");
    fl.flbrws _ TRUE;
    RETURN
    END.

(csetextname)  %Core NLS Set External Names Link File Name %
                                                                502E
    PROCEDURE( fileno, adstr );
    LOCAL TEXT POINTER stid, tps, tp1, tp2, tpe1, tpe2;
    LOCAL ldstr[40], sig2, sig3, sig4;
    REF adstr;

    stid _ origin;  stid.stfile _ fileno;  stid[1] _ 1;
    tpe1 _ adstr[ls];  tpe1[1] _ adstr[ls+1];
    tpe2 _ adstr[le];  tpe2[1] _ adstr[le+1];
    IF FIND SF(stid) ["EXTERNAL LINKS:"] $(SP/TAB/CR/LF/EOL)
    ^tps THEN
       BEGIN
       INVOKE (parselink, noenlf);
       lnkprs( $tps, $ldstr );
       DROP (parselink);
       tp1 _ ldstr[ls];  tp1[1] _ ldstr[ls+1];
       tp2 _ ldstr[le];  tp2[1] _ ldstr[le+1];
       IF tp1[1] = tps[1] THEN
          ST stid _ SF(stid) tp1, tpe1 tpe2, tp2 SE(stid)
       ELSE
          (noenlf): ST stid _ SF(stid) tps, tpe1 tpe2, tps
          SE(stid);                                          502E9H1
       END
    ELSE ST stid _ SF(stid) SE(stid), "; EXTERNAL LINKS: ",
    tpe1 tpe2;
    RETURN;

    %Define the catchphrase, parselink%

    (parselink) CATCHPHRASE (:sig2, sig3, sig4);            502E15
```

```
        BEGIN
        DISABLE (parselink);
        CASE SIGNALTYPE OF
           =aborttype: TERMINATE;
        ENDCASE CONTINUE;
        END;

     END.

(csetty)        % Core NLS Set tty window Command %
PROCEDURE ( winid );                                          502F
   LOCAL oldda REF, newda REF;
   REF dttyda, ttyda;
   % find da for window %
      &newda _ [findwa( winid )].widdarea;
   % do nothing if no change %
      IF &newda = &ttyda THEN RETURN(TRUE);
   % if the new window is the default call xrstty and return %
      IF &newda = &dttyda THEN
         BEGIN
            crstty();
            RETURN(TRUE);
         END;
   % set ttyda to the new da %
      &oldda _ &ttyda := &newda;
   % change the seq field in both da's %
      IF &oldda # &dttyda THEN oldda.daseq _ oldda.dasuppress
      _ FALSE;
      ttyda.daseq _ ttyda.dasuppress _ TRUE;
   % update the old da %
      IF &oldda # &dttyda THEN dafrmt( &oldda, 0 );
   RETURN;
   END.

(cprofil)   % Core NLS set protection of a group of files and
PCs %
PROCEDURE (rhstn, fname, mask, prot, astr % => no value %);
                                                              502G
   % Procedure description
      FUNCTION
         Sets the protection of a group of files and their
         partial copies.  (Check the TENEX JSYS Manual for
         details on file protection.)  The names of the files
         changed are put in 'astr'.
      ARGUMENTS
         rhstn - INTEGER -
            number of the host on which the files reside.
            Only the local host ('lhostn') is currently
            implemented.
         fname - STRING -
            file group name string in TENEX format.  Connected
            directory is assumed if a directory is not
            specified.  Examples:
               "*.*"
               "<SMITH>*.PC"
               "<SMITH>FOO.NLS;3"
```

```
        mask - INTEGER - mask indicating which bits to change
        prot - INTEGER - new protection bits
           77    77     77
           self  group  world
              77 = (read, write, execute, append, *, unused)
                 * = access determined by individual pages in
                 the file
           Example: 400000 = nobody but me can do anything to
           the file, and I can only read it (pretty safe!)
           Example: 777752 = the normal NLS setting
        astr - STRING - string in which to put messages
     RESULTS
        none
     NON-STANDARD CONTROL
        Error if
           host is not the local host
           a file doesn't exist or can't be changed
           not a disk file
     GLOBALS
        <lots>
     EXAMPLE
        cprofil(lhostn, $"<SMITH>*.NLS", 777777B, 775200B,
        $string)
           Meaning: I can do anything, my group can read and
           execute, others can do nothing (not even get a
           directory listing of it).
     %
  % Declarations %
     LOCAL
        fjfn, % main jfn %
        jfnflgs,      % left half flags for a group jfn %
        pcjfn,        % jfn of partial copy %
        flags;        % bits indicating * typed for file name
        fields %
     LOCAL STRING
        uname[200],           % complete name as entered by
        user %
        udirname[40], % user input directory name %
        ufilname[40], % user input file name %
        uextname[40], % user input extension name %
        uverno[40],           % user input version number %
        ulftovr[40],  % user input beyond version number %
        jfnname[200], % complete name from jfns %
        errstring[200],       % eror message string %
        tstring[200]; % temporary string %
     REF fname, astr;
  % initial variables %
     fjfn _ jfnflgs _ pcjfn _ flags _ 0;
  % find out if remote or local (disk) file %
     CASE rhstn OF
        = lhostn:  NULL;
        ENDCASE err( $"remote file manipulations not
        implemented yet" );
  % now parse user input strings %
     parseinput( &fname, $udirname, $ufilname, $uextname,
     $uverno, $ulftovr, $uname, $flags);
```

```
% get main jfn (old file only, group jfn) %
   IF NOT (fjfn _ sgtjfn( gtjoif .V gtjstr + 0, $uname,
   $errstring : jfnflgs ) )THEN err( $errstring );
% now find out if we support this type of file %
   IF NOT chkdev( fjfn ) THEN err( $"only disk files
   supported" );
% now loop to get all files in the group %
   LOOP
      BEGIN
      % check to make sure this not a PC with * for
      extension name %
         IF NOT chkpcs( fjfn, flags) THEN GOTO gpronxjfn;
      % get actual file name into string for use later
      (maybe) %
         R3 _ 011110B6      %  directory file ext ver %
            + (IF jfnflgs.lhjb9  THEN 1B6 ELSE 0) %
            ;Protection %
            + (IF jfnflgs.lhjb10 THEN 1B5 ELSE 0) %
            ;Account %
            + (IF jfnflgs.lhjb11 THEN 4B4 ELSE 0) % ;T %
            + 1;              % with proper file punctuation
            %
         jfnflink( fjfn, $jfnname, R3);
      % get PC jfn and find out if this user can access
      this PC %
         IF NOT getpcjfn( fjfn, $pcjfn, $errstring, FALSE)
         THEN
            BEGIN   % this user can't access this file %
            *tstring* _ *jfnname*, " protection not changed
            because ", *errstring*;
            IF flags THEN
               BEGIN
               *astr* _ *astr*, "   ***   ", *tstring*, CR,
               LF;
               GOTO gpronxjfn;
               END
            ELSE
               BEGIN
               IF NOT SKIP !rljfn( fjfn) THEN NULL;
               err( $tstring );
               END;
            END;
      % now protect the file (and its partial copy) %
         IF NOT proflandpc( fjfn, pcjfn, mask, prot,
         $errstring) THEN
            BEGIN   % can't delete file (or partial copy) %
            *tstring* _ *jfnname*, " protection not changed
            because ", *errstring*;
            IF flags THEN
               BEGIN
               *astr* _ *astr*, "   ***   ", *tstring*, CR,
               LF;
               GOTO gpronxjfn;
               END
            ELSE
               BEGIN
```

```
                    IF NOT SKIP !rljfn( fjfn) THEN NULL;
                    IF NOT SKIP !rljfn( pcjfn) THEN NULL;
                    err( $tstring );
                    END;
                END;
            % now tell the user we have protected this file %
              *astr* _ *astr*, "   ", *jfnname*;
              IF pcjfn THEN *astr* _ *astr*, " and its partial
              copy";
              *astr* _ *astr*, CR, LF;
            % now get rid of the jfn for the partial %
              IF NOT SKIP !rljfn( pcjfn ) THEN NULL;
            % now get the next file in the group %
              (gpronxjfn):                                    502G8A8A
              R1.LH _ jfnflgs;
              R1.RH _ fjfn;
              IF NOT SKIP !gnjfn( R1 ) THEN EXIT LOOP;
            END;
        % now get rid of any lingering jfns %
            IF NOT SKIP !rljfn( fjfn ) THEN NULL;
            IF NOT SKIP !rljfn( pcjfn ) THEN NULL;
        % all done now, so return %
           RETURN;
        END.   %%
```

```
%show%
   %show PCP interface routine%
      (xshow)   % CL: ;  Execute show Command %
      PROCEDURE (entity REF, type REF, dopt REF, window, rtnlist
      REF);                                                    5P1A
         % Procedure description
            FUNCTION
               none
            ARGUMENTS
               entity:        entity type
               type:          further entity type specification
               dopt:          list of directory options
               window:        window to be shown status for
               rtnlist:       return arg
               none
            RESULTS
               proc-value
            NON-STANDARD CONTROL
               none
            GLOBALS
               dsparg:  sets and nulls oute
            %
         % Declarations %
            LOCAL
               rhostn, retry, dskcnt, destination REF, statusstr
               REF, enttype, i, diraddr,
               % stuff for show directory %
                  info,       % record saying what was requested %
                  gropk,      % record saying how to group things %
                  sortk;      % record saying how to sort things %
            LOCAL STRING filstr[200], badmsg[200];
            LOCAL LIST optlist[10];
            REF dsparg;
         %Invoke catchphrase%
            INVOKE (catxshow);
         *lit* _ NULL;  %used for building status messages%
         CASE  (enttype _ ELEM #entity#[cwtype]) OF
            = 9 %- directory -%:
               BEGIN
               info _ gropk _ sortk _ 0;
               *filstr* _ *dirsfname*;
               % convert directory option list %
                  IF &dopt THEN
                     FOR i _ 1 UP UNTIL > dopt.L DO
                        cnvarg(&dopt, $optlist, i);
               % check for NULL directory name %
                  diraddr _ IF &type THEN ELEM #type# [tppair] ELSE
                  0;
               xdiropt(diraddr, $optlist, $info,
                  $gropk, $sortk, $rhostn, $filstr);
               cshodir(info, gropk, sortk, rhostn, $filstr);
               *lit* _ " ";  % output already done, grammar
               specifies final CONFIRM %
               END;
            = 52 %- disk -%: %space status% %uses connected
```

```
            directory%
               IF (dskcnt _ cshodskspa($lit)) > 0 THEN
                  BEGIN
                  !gjinf();
                  gdname(R2, $filstr);
                  *filstr* _
                     *filstr*, " OVER ALLOCATION BY ",
                     STRING(dskcnt), " PAGES!";
                  dismes(2, $filstr);
                  END;
            = 51 %- file -%: %status%
               CASE ELEM #type#[cwtype] OF
                  = 52 %- status -%: %all%
                     cshofilsta(7, dspfile(window), $lit);
                  =  54 %- default -%: %dir for links%
                     cshofilsta(4, dspfile(window), $lit);
                  = 51 %- marker -%: %list%
                     cshomarfil(dspfile(window), $lit);
                  = 53 %- modifications -%: %status%
                     cshofilsta(2, dspfile(window), $lit);
                  = 62 %- return -%: %ring%
                     cshofrring(dsparea(window), $lit);
                  = 55 %- size -%:
                     cshofilsta(1, dspfile(window), $lit);
                  ENDCASE
                  IF NOT donthelp THEN
                     BEGIN
                     *badmsg* _ "The file status type argument was
                     incorrect.  The allowed values are ";
                     *badmsg* _ *badmsg*, "Default, Marker,
                     Modifications, Return, Size, or Status.";
                     retry _ HELP (badarg, $badmsg, cindex);
                     REPEAT CASE (retry);
                     END
                  ELSE ABORT (badarg, $"The property type argument
                  was incorrect.");
            = 62 %- return -%: %ring status%
               cshosrring( dsparea(window), $lit );
            = 32 %- name -%: %delimiters for statement%
               BEGIN
               &destination _ ELEM #type#[tppair];
               cshonsta(destination, $lit);
               END;
            = 57 %- viewspecs -%:
               cshoviespe(dsparea(window), $lit);
            = 58 %- buffer -%:
               BEGIN
               &statusstr _ getstring(200, lstzone);  %allocate in
               list zone%
               gpstatus (&statusstr, dsparea(window));
               #dsparg# _ USE makedesc(ustring, &statusstr, TRUE),
               USE makedesc(uboolean, cawait, FALSE);
               extcall (feident, $"show", fedpypkg, &dsparg);
               END;
            ENDCASE
            IF NOT donthelp THEN
```

```
            BEGIN
            retry _ HELP (badarg, $"The type argument was
            incorrect.  The allowed values are Buffer, Disk,
            File, Name, Options, Return, or Viewspecs.", cindex);
            REPEAT CASE (retry);
            END
         ELSE ABORT (badarg, $"The type argument was
         incorrect.");
      IF NOT nodisplay THEN dpset(dspno, endfil, endfil, endfil);
      #rtnlist#[1] _ USE rtnstring($lit);
      % Drop catchpharse and free space %
         DROP (catxshow);
         IF dsparg.L THEN #dsparg# _ ;
         #optlist# _ ;
      % Return %
         RETURN;
      % Catchphrase %
         (catxshow) CATCHPHRASE();                              5P1A10A
            BEGIN
            CASE SIGNALTYPE OF
               = notetype :
                  CASE SIGNAL OF
                     = return, = unwind:
                        BEGIN  %null list%
                        DISABLE (catxshow);
                        #dsparg# _ ;
                        #optlist# _ ;
                        END;
                     ENDCASE;
               = helptype :
               = aborttype :
               ENDCASE;
            CONTINUE;
            END;
      END.

%show core routines%
   (cshoviespe)  PROCEDURE (da, string);                        5P2A
      LOCAL vs[2];
      REF da, string;
      vs _ da.davspec;
      vs[1] _ da.davspc2;
      curvsp($vs, &string);
      RETURN;
      END.

   (cshodskspa)   %Core NLS Show Disk Space Status Command%     5P2B
      PROCEDURE   %uses connected directory%
         (astr);  % address of string to receive message %     5P2B1A
      LOCAL
         fjfn,    % main jfn %
         jfnflgs, % left half flags for a group jfn %
         size,    % size (in pages) of an individual file %
         usrinu,  % in use pages this directory %
         usrall,  % allowed pages this directory %
         usrund,  % undeleted pages this directory %
```

```
        usrdel,  % deleted pages this directory %
        sysinu,  % used pages in system %
        sysfre   % free pages in system %
        ;
    LOCAL STRING
        dirname[40],     % string to get directory name %
        errstring[200]   % eror message string %
        ;
    REF astr;

    % initial variables %
        fjfn _ jfnflgs _ usrinu _ usrall _ usrund _ usrdel _
        sysinu _ sysfre _ 0;
    % get main jfn (old file only, group jfn) %
        IF NOT (fjfn _ sgtjfn( gtjoif .V gtjstr .V gtjidl,
        $dirsfname, $errstring : jfnflgs ) )THEN
            BEGIN
            IF (R1.RH = gjfx20) OR (R1.RH = gjfx32) THEN NULL
            ELSE ABORT(0, $errstring );
            END;
    % loop to count usage for this directory %
        IF fjfn THEN LOOP
            BEGIN
            % get number of pages this file %
              !gtfdb( fjfn, 1000011B, $R3);
              size _ R3.RH;
            % add to total in use count %
              usrinu _ usrinu + size;
            % get deletion status of this file %
              !gtfdb( fjfn, 1000001B, $R3);
            % now update the appropriate sum %
              IF R3.fdbdel THEN usrdel _ usrdel + size
              ELSE usrund _ usrund + size;
            % now get the next file in the group %
              (gndsjfn):                                    5P2B8A6A
              R1.LH _ jfnflgs;
              R1.RH _ fjfn;
              IF NOT SKIP !gnjfn( R1 ) THEN EXIT LOOP;
            END;
    % now get rid of any lingering jfns %
         IF NOT SKIP !rljfn( fjfn ) THEN NULL;
    % get total allowed for this directory %
        !gtdal(IF tops20flag THEN -1 ELSE 0 );
        usrall _ R1;
    % get system totals %
        !gdskc( 600000777777B );
        sysinu _ R1;
        sysfre _ R2;
    % get connected directory name %
        !gjinf();
        gdname( R2.RH, $dirname );
    % now build the message string %
        *astr* _ "Connected to ", *dirname*, EOL,
        STRING(usrinu), " Total pages in use -- ",
        STRING(usrall), " Allowed, ", STRING(usrund), "
        Undeleted, ", STRING(usrdel), " Deleted", EOL, "System
```

```
      Total: ", STRING(sysfre), " Pages left, ",
      STRING(sysinu), " Used";
   % all done now, so return %
      RETURN(usrinu - usrall);
   END.

(cshofilsta)          %Core NLS Show Status of File%        5P2C
   PROCEDURE(type, fileno, astrng);
   REF astrng;
      %type is ALL, link, modifications, size%
   fstatus(fileno, &astrng, type);
   RETURN;
   END.

(cshofrring)   %Core NLS Show Status of File Return Ring%  5P2D
   PROCEDURE (da, astrng);
   LOCAL end, i, jfn, filstr, fileno, flags;
   LOCAL TEXT POINTER t1, t2, u1, u2, t1, f2, n1, n2, v1, v2;
   LOCAL STRING locstr[200];
   REF da, astrng;
   *astrng* _ NULL;
   IF NOT da.daexis THEN
      ABORT(badda,$"Illegal display area address in
      cshofrring");
   end _ frrlength(da.dalink);
   INVOKE (repeatloop, rptloop);  %in case top entry
   non-existant%
   DISABLE (repeatloop);
   FOR i _ 0 UP UNTIL > end DO
      BEGIN
      ENABLE (repeatloop);
      filstr _ readfrring(da.dalink, i);
      DISABLE (repeatloop);
      FIND SF(*[filstr]*) ^t1;
      lnbfls( $t1, 0, $locstr);
      %+NSW%                                                5P2D11G
         %No lock message for now.  Could get project.node of
         the PC ownerner by calling wmreadsem.%
         *astrng* _ *astrng*, *[filstr]*, EOL;
      %+NSW%                                                5P2D11H
      %-NSW%                                                5P2D11I
      fileno _ filnum($locstr);
      IF  fileno  = -1 THEN
         BEGIN %file not open -- get a jfn%
         flags _ getgtjflg(write, origff, oldvrsn);
         jfn _ sgtjfn(flags, $locstr, $locstr);
         END
       ELSE jfn _ [flntadr(fileno)].florig;
      R3 _ 0;
      IF jfn THEN
         BEGIN
         *locstr* _ NULL;
         !gtfdb(jfn, 1000024B, $R3);
         END;
      IF R3.lkinit THEN
         lockid($locstr, R3.lkdirn, R3.lkinit);
```

```
        *astrng* _ *astrng*, *[filstr]*, *locstr*, EOL;
        IF fileno = -1 THEN reljfn(jfn);
        %-NSW%                                                    5P2D11R
        (rptloop);                                                5P2D11S
        END;
    DROP (repeatloop);
    RETURN;
    (repeatloop) CATCHPHRASE();                                   5P2D14
        BEGIN
        CASE SIGNALTYPE OF
            = notetype :  NULL;
            = helptype :  NULL;
            = aborttype :
                IF SIGNAL = frremptyentry THEN TERMINATE;
            ENDCASE;
        DISABLE (repeatloop);
        CONTINUE;
        END;
    END.

(cshosrring) PROCEDURE % show statement return ring %             5P2E
    % FORMAL ARGUMENTS %
        (da, astr);
    LOCAL stid, stdb, len, cc, pvs1, pvs2, srr, i;
    LOCAL TEXT POINTER tp1,tp2;
    LOCAL STRING temp[70];
    REF da, srr, astr;
    % -------------- %
    % advance through jump stack %
        INVOKE (badfrr);
        readfrring(da.dalink, 0 : &srr);
        DROP (badfrr);
        srrcnt _ srrlength( &srr );
    FOR i _ 1 UP UNTIL >= srrcnt DO
        BEGIN
        stid _ readsrring(&srr, i : cc, pvs1, pvs2);
        %display the current statement%
            % display first n chars of stmt in name area %
                % get statement length %
                    IF NOT lodprop( stid, txttyp : stdb) THEN
                        err($"No text block associated with node");
                    len _ [stdb].schars + 1; % number of chars %
                % construct text ptrs to each end of stmt %
                    tp1 _ stid;
                    tp1[1] _ 1;
                    tp2 _ stid;
                    %truncate if over n chars %
                        tp2[1] _ MIN(IF nlmode = fulldisplay THEN 60
                        ELSE 20,len);
                % assign something to the string "temp" %
                    IF len > 1 THEN *temp* _ tp1 tp2 ELSE *temp* _
                    "<NULL>";
        *astr* _ *astr*, *temp*, EOL;
        END;
    RETURN;
```

```
   (badfrr) CATCHPHRASE();                                          5P2E10
      BEGIN
      DISABLE (badfrr);
      CASE SIGNALTYPE OF
         = notetype :  NULL;
         = helptype :  NULL;
         = aborttype :
            IF SIGNAL = frremptyentry THEN
               err($"No current entry in file return ring.");
         ENDCASE;
      CONTINUE;
      END;
   END.

(cshomarfil)   %Core NLS Show Markers in File Command%      5P2F
   PROCEDURE(fileno, astr);
   REF astr;
   seemkr(fileno, &astr);
   RETURN;
   END.

(cshonsta) PROCEDURE %***%  %Core NLS Show Name Delimiters for
Statement %                                                      5P2G
   (stid, astrng);                                               5P2G1
   LOCAL dlleft, dlrght;
   REF astrng;
   dlleft _ getnmdl( stid : dlrght);
   IF dlleft = 0 THEN *astrng* _ "NULL", SP
   ELSE *astrng* _ dlleft, SP;
   IF dlrght = 0 THEN *astrng* _ *astrng*, "NULL"
   ELSE *astrng* _ *astrng*, dlrght;
   RETURN;
   END.

(cshodir)   % GB: core NLS Show Directory procedure %
PROCEDURE (info, gropk, sortk, rhstn, fname % => no value %);
                                                                 5P2H
   % Procedure description
      FUNCTION
         Prints the names of the files specified by 'fname'
         into the literal display area.
      ARGUMENTS
         info - DLINFO - record describing what to list, or 0
         gropk - DLGRP - record containing grouping
         information, or 0
         sortk - DLSORT - record containing sorting
         information, or 0
         rhstn - INTEGER -
            number of the host on which the files reside.
            Only the local host ('lhostn') is currently
            implemented.
         fname - STRING -
            file group name string in TENEX format.  Connected
            directory is assumed if a directory is not
            specified.  Examples:
               "*.*"
```

```
                "<SMITH>*.PC"
                "<SMITH>FOO.NLS;3"
        RESULTS
           none
        NON-STANDARD CONTROL
           Error if
              host is not the local host
              files are illegal or not disk files
              usual TENEX file system errors
              various internal errors
        GLOBALS
           <lots>
        EXAMPLE
           cshodir(0, 0, 0, lhostn, $"*.NLS;*")
        %
     % Declarations %
        LOCAL
           noverf,         % no version numbers flag %
           noextf,         % no extension names flag %
           sortdi,                 % sort direction %
           tptr,           % temp pointer for building final
           string %
           chns,           % pointer to data structure %
           adlmb,          % address of directory list storage %
           fjfn,           % main jfn %
           jfnflgs,        % left half flags for a group jfn %
           flags           % bits indicating * typed for file name
           fields %
           ;
        LOCAL TEXT POINTER
           tp1, tp2, tp3;
        LOCAL STRING
           nwname[200],  % current file name link %
           oldname[200], % previous file name link %
           astr[2000],   % string for listing a file %
           uname[200],   % complete name as entered by user %
           udirname[40], % user input directory name %
           ufilname[40], % user input file name %
           uextname[40], % user input extension name %
           uverno[40],   % user input version number %
           ulftovr[40],  % user input beyond version number %
           jfnname[200], % complete name from jfns %
           errstring[200],       % eror message string %
           tstring[200]  % temporary string %
           ;
        REF fname, tptr, chns, adlmb;
     % initial variables %
        fjfn _ jfnflgs _ flags _ &tptr _ &chns _ &adlmb _ 0;
        sortdi _ sortk.dlsrvr := FALSE;
        noverf _ info.dlinvr := FALSE;
        noextf _ info.dlinex := FALSE;
     % find out if remote or local (disk) file %
        CASE rhstn OF
           = lhostn:  NULL;
           ENDCASE err( $"remote file manipulations not
           implemented yet" );
```

```
% now parse user input strings %
   parseinput( &fname, $udirname, $ufilname, $uextname,
   $uverno, $ulftovr, $uname, $flags);
% get main jfn (old file only, group jfn) %
   IF info.dlidlt THEN
      BEGIN
      IF NOT
         (fjfn _
            sgtjfn(
               gtjoif .V gtjstr .V gtjidl,
               $uname, $errstring : jfnflgs
            )
         )
            THEN err( $errstring );
      END
   ELSE
      IF NOT
         (fjfn _
            sgtjfn(
               gtjoif .V gtjstr, $uname, $errstring :
               jfnflgs
            )
         )
            THEN err( $errstring );
% now find out if we support this type of file %
   IF NOT chkdev( fjfn ) THEN err( $"only disk files
   supported" );
% get directory list storage %
   IF NOT (&adlmb _ dlgmb(400)) THEN
      err($"Internal Storage Problem");
% cleanup on any problems %
   INVOKE(cshodcat);
% go create the data structure %
   IF NOT  (&chns _
      dldriver(info, gropk, sortk, fjfn, jfnflgs,
      $errstring, &adlmb)) THEN
         BEGIN
         IF NOT SKIP !rljfn( fjfn ) THEN NULL;
         dlfmb( &adlmb := 0);
         IF errstring.L THEN err($errstring)
         ELSE err($"System Screwup");
         END;
% broadcast any error message %
   IF errstring.L THEN dismes(2,$errstring);
% get ready to list the directory %
% build string for one file, add to list string,  and loop
for all files %
   WHILE &chns DO
      BEGIN
      IF inptrf THEN EXIT LOOP;
      IF sortdi THEN &tptr _ chns.dlgbnu
      ELSE &tptr _ chns.dlgbsc;
      WHILE &tptr DO
         BEGIN
         IF inptrf THEN EXIT LOOP 2;
         IF NOT tptr.dlfbfl.dlstig THEN
```

```
                BEGIN
                astr.L _ 0;
                IF noverf OR noextf THEN
                   BEGIN
                   FIND
                      SE(*[tptr.dlfbal]*) [",] ^tp3 [";  / ".]
                      ^tp2 [".] ^tp1;
                   IF noverf AND noextf THEN
                      *nwname* _ SF(*[tptr.dlfbal]*) tp1,
                         tp3 SE(*[tptr.dlfbal]*)
                   ELSE
                      IF noverf THEN
                         *nwname* _ SF(*[tptr.dlfbal]*) tp2,
                            tp3 SE(*[tptr.dlfbal]*)
                      ELSE
                         *nwname* _ SF(*[tptr.dlfbal]*) tp1,
                            tp2 SE(*[tptr.dlfbal]*);
                   IF *nwname* = *oldname* THEN GOTO sbypass
                   ELSE *oldname* _ *nwname*;
                   *astr* _ *astr*, *nwname*;
                   END
                ELSE
                   *astr* _ *astr*, *[tptr.dlfbal]*;
                % build additional line 1 information string %
                   dlbldai(
                      fjfn, tptr.dlfbpf, tptr.dlfbff,
                      tptr.dlfbpf, $astr);
                *astr* _ *astr*, CR, LF;
                % build information string for this file %
                   dlbldi(
                      info, tptr.dlfbpf, tptr.dlfbff,
                      tptr.dlfbpf, $"   ", $astr);
                % now list this file %
                   xdlpnt( $astr );
                END;
             (sbypass):                                          5P2H13A5D
             IF sortdi THEN &tptr _ tptr.dlfbpb
             ELSE &tptr _ tptr.dlfbnb;
             END;
          &chns _ chns.dlgbnb;
          END;
       % get rid of any lingering storage %
          dlfmb( &adlmb := 0 );
       % make sure we get rid of any jfns %
          IF NOT SKIP !rljfn( fjfn ) THEN NULL;
       % send string of file info %
          xdlpnt(FALSE);
       % drop catchphrases %
          DROP(cshodcat);
       % return %
          RETURN;
       % catchphrase %
          (cshodcat) CATCHPHRASE();                            5P2H19A
             BEGIN
             CASE SIGNALTYPE OF
                = notetype :
```

```
                = helptype :
                = aborttype :
                   BEGIN
                   DISABLE(cshodcat);
                   dlfmb(&adlmb := 0);
                   IF NOT SKIP !rljfn( fjfn ) THEN NULL;
                   END;
                ENDCASE;
             CONTINUE;
             END;
      END.   %%
```

```
      % utility for show directory - prints one file %
         (xdlpnt)  % print a file for show directory %          5P2I1
            PROCEDURE
               (astr REF     % addr of string to send to FE command
               feedback window, or FALSE (meaning send what you
               have) %
               );
            IF &astr = 0 OR (lit.L + astr.L > lit.M) THEN
               BEGIN
               typelit(0, $lit); % let FE handle confirms at end of
               window %
               lit.L _ 0;
               END;
            IF &astr THEN *lit* _ *lit*, *astr*;

            RETURN;

            END.

%simulate%
   %simulate PCP interface routine%
      (xsimulate)    % simulate tty/display xroutine %
      PROCEDURE( type REF );                                    5Q1A
         LOCAL da REF;
         &da _ lda();
         CASE ELEM #type#[1] OF
            = dsply:
               BEGIN
                  % [tda].daseq _ FALSE; %
                  da.daseq _ FALSE;
                  nlmode _ fulldisplay;
                  simtty _ FALSE;
               END;
            = typwrtr:
               BEGIN
                  % [tda].daseq _ TRUE; %
                  da.daseq _ TRUE;
                  nlmode _ typewriter;
                  simtty _ TRUE;
               END;
         ENDCASE;
         RETURN( TRUE );
         END.

      (xchksimtty)   % check if tty is simulated %
      PROCEDURE(result REF );                                   5Q1B
         % Procedure description
            FUNCTION
               Checks and returns the value of simtty to the FE
            ARGUMENTS
               none
            RESULTS
               TRUE -- If this is a display that simulates a tty
               FALSE -- No simulation in effect
            NON-STANDARD CONTROL
```

```
            none
         GLOBALS
            none
         %
      #result# _ USE makedesc(uboole, simtty, FALSE);
      RETURN( &result );
      END.

   %simulate core routine%
%sort%
   %sort PCP interface routine%
      (xsort) %Execute Sort Command%                                    5R1A
         PROCEDURE (
            %FORMALS%
               destptr REF,  %destination pointer%
               window        %window number%
               );
         LOCAL retry REF, destination REF;
         %----------------------%
         IF &destptr THEN &destination _ ELEM #destptr#[tppair];
         CASE ELEM #destptr#[cwtype] OF
            = 27 %- group -%, = 28 %- plex -%:
               BEGIN
               sortgrp (destination, destination[d2sel], window);
               curmkr _ gethed(destination);  curmkr[1] _ 1;
               IF NOT nodisplay THEN dpset(dspstrc, destination,
               endfil, endfil);
               END;
            = 26 %- branch -%:
               BEGIN
               IF (destination := getsub(destination)) = destination
               THEN ABORT(badsort, $"Illegal sort.");
               destination[d2sel] _ getail(destination);
               REPEAT CASE (27 %+ group +% );
               END;
            ENDCASE
            IF NOT donthelp THEN
               BEGIN
               &retry _ HELP (badarg, $"The selection type was
               incorrect.  The allowed values are Branch, Group, or
               Plex.", cindex);
               destination _ ELEM #retry#[tppair];
               REPEAT CASE (ELEM #retry#[cwtype]);
               END
            ELSE ABORT (badarg, $"The selection type was
            incorrect.");
         RETURN;
         END.
%substitute%
   %substitute PCP interface routine%
      (xsubstitute) %Execute Substitute Command% PROCEDURE (           5S1A
         %FORMALS%
            entity REF,       %entity type %
            destptr REF,                %destination pointer%
            pairslist REF,    %address of pairs list%
            vs REF            %viewspecs for filter%
```

```
    );
LOCAL wsave, retry REF, destination REF, pairs, save1,
save2, savca, savus, da REF, adstr[40], sig2, sig3, sig4,
initblk[2], i, textent, tp1, tp2;
LOCAL LIST pairsptr[10];
%----------------------%
IF &destptr THEN &destination _ ELEM #destptr#[tppair];
textent _ ELEM #entity# [cwtype];
%Initialize the state of the substitute pair record; this
must be done before calls on subpsave%
   subsinit (TRUE, $textent, $initblk);  %I hope this (arg
   2) gets the type of the first selection%
   INVOKE (subfree);
%Convert to Backend format%
   wsave _ cwndow;  %cnvarg may change current window
   global%
   % we must do argument conversion here since argument
   conversion does not convert lists.  conversion must be
   done since there may be instances in which the string is
   not in the list (e.g., an address was specified.  we
   then convert again to a data structure suitable for use
   by the (old) substitute core routines. %
      FOR i _ 1 UP 2 UNTIL > pairslist.L DO
         BEGIN
         cnvarg (&pairslist, $pairsptr, i);
         cnvarg (&pairslist, $pairsptr, i+1);
         tp1 _ ELEM #[ELEM #pairsptr#[i]]# [tppair];
         tp2 _ ELEM #[ELEM #pairsptr#[i+1]]# [tppair];
         pairs _ subpsave (tp1, tp2);
         END;
   cwrdow _ wsave;
&da _ dsparea(ELEM #destptr#[wndw]);
curmkr _ destination;  curmkr[1] _ 1;
save1 _ da.davspec;
save2 _ da.davspc2;
savca _ da.dacacode;
savus _ da.dausqcod;
INVOKE (resetda);
% This code should be replaced by a core substitue
procedure and a call on fltprc %
IF &vs THEN
   BEGIN
   da.davspec _ vs.vs1;
   da.davspc2 _ vs.vs2;
   da.dacacode _ vs.vscacode;
   da.dausqcod _ vs.vsusqcod;
   END
ELSE
   BEGIN
   da.davspec _ 0;
   da.davspec.vslev _ da.davspec.vstrnc _
      da.davspec.vsnamf _ da.davspec.vsdaft _
         da.davspec.vsindf _ -1;
   da.dacacode _ da.dausqcod _ 0;
   END;
CASE ELEM #destptr#[cwtype] OF
```

```
        = 29 %- statement -%:
           BEGIN
           IF NOT nortnrings THEN
              clist(ctcmk, destination.stfile, nofile);
           IF NOT nodisplay THEN
              dpset(dsprfmt, destination, endfil, destination);
           csubsta(destination, pairs, &da);
           IF NOT nortnrings THEN clupdt();
           END;
        = 27 %- group -%, = 28 %- plex -%, = 26 %- branch -%:
           BEGIN
           IF NOT nortnrings THEN
              clist(ctcmk, destination.stfile, nofile);
           IF NOT nodisplay THEN
              dpset(dspallf,destination,endfil,endfil);
           csubgro(destination, [&destination+d2sel], pairs,
           &da);
           IF NOT nortnrings THEN clupdt();
           END;
        ENDCASE
           IF NOT donthelp THEN
              BEGIN
              &retry _ HELP (badarg, $"The selection type was
              incorrect.  The allowed values are Branch, Group,
              Plex, or Statement.", cindex);
              destination _ ELEM #retry#[tppair];
              REPEAT CASE (ELEM #retry#[cwtype]);
              END
           ELSE ABORT (badarg, $"The selection type was
           incorrect.");
     subsinit (FALSE, 0, $initblk); % frees substitute work area
     %
     DROP (subfree);
     da.davspec _ save1;
     da.davspc2 _ save2;
     da.dacacode _ savca;
     da.dausgcod _ savus;
     DROP (resetda);
     #pairsptr# _ 1;
     RETURN;

     %Define catchphrases%

     (subfree) CATCHPHRASE;                                    5S1A32
        BEGIN
        DISABLE (subfree);
        CASE SIGNALTYPE OF
           =aborttype:
              BEGIN
              subsinit (FALSE, 0, $initblk); % frees substitute
              work area %
              END;
           ENDCASE;
        END;
     (resetda) CATCHPHRASE;                                    5S1A33
        BEGIN
```

```
        DISABLE (resetda);
        CASE SIGNALTYPE OF
          =aborttype:
            BEGIN
            #pairsptr# _ ;
            da.davspec _ save1;
            da.davspc2 _ save2;
            da.dacacode _ savca;
            da.dausqcod _ savus;
            END;
          ENDCASE;
        END;
      END.

(xsubresolve)   % CL: ;  resolve entities in substitute list %
PROCEDURE (pairslist REF, retlist REF);                    5S1B
  % Procedure description
    FUNCTION
      Form strings for entities in substitute list when
      status is requested.  This routine is called only if
      the FE cannot resolve the entities, i.e. for address
      expressions or entities that are not known in the FE
      (e.g. NUMBER).
    ARGUMENTS
      pairslist: address of list of pairs of new and old
      entities from FE
      retlist: address of list of pairs of new and old
      entities converted to strings
    RESULTS
      proc-value
    NON-STANDARD CONTROL
      none
    GLOBALS
      none
    %
  % Declarations %
    LOCAL wsave, i, ptr1 REF, ptr2 REF, listptr REF;
    LOCAL LIST pairsptr[10];
    LOCAL STRING string[200];
  % Create list big enough to hold all the strings %
    &listptr _ getlst(pairslist.L);
    #listptr# _ ;
  % Convert each entity to a string %
    wsave _ cwndow;  %cnvarg may change current window%
    FOR i _ 1 UP UNTIL > pairslist.L DO
      BEGIN
      %Convert to backend format%
        cnvarg (&pairslist, $pairsptr, i);
      %Form string%
        &ptr1 _ ELEM #[ELEM #pairsptr#[i]]# rtppair];
        &ptr2 _ &ptr1 + d2sel;
        *string* _ ptr1 ptr2;
      %Add element to list%
        #listptr# !_ *string*;
      END;
    cwndow _ wsave;  %restore current window global%
```

```
        % Place list of strings in return list %
           #retlist# _ USE makedesc (ulist, &listptr, TRUE);
        % Return %
           RETURN;
        END.

   %substitute core routine%
      (csubgro)                %Core-NLS substitute group%          5S2A
        PROCEDURE (stid1, stid2, sbheadb, da);
        LOCAL stid, vspec,
           sw;   % address of sequence work area %
        REF da, sbheadb, sw;
        %---------%
        IF &da NOT IN [$dpyarea, $dpyarea + dal*dacnt) OR
           NOT da.daexis OR da.daempty THEN ABORT(badsub,$"Illegal
           Substitute");
          %msntx();%
        dismes (2, $"Substitute in Progress");
        stid1 _ grptst(stid1, stid2 : stid2);
        vspec _ da.davspec;
        vspec.vsbrof _ vspec.vsplxf _ FALSE;
        &sw _ openseq (stid1, stid2, vspec, da.davspc2,
        da.dausgcod, da.dacacode);
        WHILE (stid _ seggen (&sw)) # endfil DO csubsta(stid,
        &sbheadb, &da);
        closeseq (&sw);
        dismes(0);
          %msftx();%
        RETURN;
        END.

      (csubsta) %***%                %Core-NLS substitute statement%
                                                              5S2B
        PROCEDURE (stid, sbheadb, da);
        LOCAL
           cary,   %pointer (later SKIPN) to subdsp%
           ttype,  %type of text entity being substituted%
           sfc,    %first char position for scan%
           sbp,    %byte pointer to chars in statement%
           sfbp,   %initial byte pointer in statement%
           lbp,    %byte pointer to left delimiter%
           rbp,    %byte pointer to right delimiter%
           snc,    %counter for chars in statement%
           ch,     %last char read from statement%
           chbp,   %byte ptr to end of last change%
           chnc,   %char pos of last change from right end of
           original statement%
           chflag, %count of changes made%
           cap,    %ptr to substitution description record%
           cbp,    %byte ptr to candidate or replacement%
           cnc,    %length of candidate or replacement%
           cnc1,   %length of candidate%
           tbp,    %temp sbp for comparison%
           wbp,    %byte ptr to last char written in new statement%
           wnc,    %number of chars written in new statement%
           maxsl;  %max size of new statement%
```

```
    POINTER sbheadb, da, cap;
      %msntx();%
    cary _ sbheadb.sbdp;
    ttype _ sbheadb.sbtt;
    sfc _ IF stid.stastr OR da.davspec.vsnamf THEN 1 ELSE
    fchtxt(stid);
    sfbp _ sbp _ stbpget(stid, sfc: snc);
       snc _ snc + 1 - sfc;
    IF sfc = 1 THEN BEGIN %entire statement%
       chbp _ sbp; chnc _ snc;
    END ELSE %name not scanned%
       chbp _ stbpget(stid, 1: chnc);
    chflag _ 0;
    maxsl _ lit.M;
    wbp _ 0107B8 + $lit; %text begins at $lit+1%
    wnc _ 0;
    % set up for fast scan %
       A1 _ skipni;
       !HLLM A1,cary; %insert instruction part%
    (step1):                                                    5S2B16
       BUMP snc;
    (step): %fast scan loop%                                    5S2B17
       !SOSG snc;
       !JRST done;
       !ILDB A1,sbp;
       !XCT cary; %SKIPN A2,subdsp(A1)%
       !JRST step;
       cap _ A2;
       DO
          BEGIN
          IF (cnc _ (cnc1 _ cap.catnc)-1) < snc THEN
             BEGIN
             tbp _ sbp;
             cbp _ cap.catbp;
             ch _ ^cbp; %skip first char%
             FOR cnc DOWN UNTIL <=0 DO
                IF ^tbp # ^cbp THEN GOTO noteq;
             % found match, check delimiters %
             IF (lbp _ bkbp(sbp)) # sfbp THEN %check left
             delimiter%
                IF NOT sbdelck(.lbp,ttype) THEN GOTO noteq;
             IF snc # cnc1 THEN %check right delimiter%
                BEGIN
                rbp _ tbp;
                IF NOT sbdelck(^rbp,ttype) THEN GOTO noteq;
                END;
             IF (wnc _ wnc + chnc - snc + (cnc _ cap.carnc))
                > maxsl THEN
                   BEGIN
                   % generate string system overflow error %
                   lit.L _ lit.M;
                   *lit* _ *lit*, SP;
                   RETURN; %if no trap - may not be possible%
                   END;
             % copy segment between last match and this %
             UNTIL chbp = lbp DO
```

```
                    ^wbp _ ^chbp;
                % enter replacement %
                cbp _ cap.carbp;
                FOR cnc DOWN UNTIL <=0 DO
                    ^wbp _ ^cbp;
                % update variables %
                chbp _ sbp _ tbp;
                chnc _ snc _ snc - cnc1;
                BUMP chflag;
                GOTO step1;
             END;
             (noteq):                                          5S2B17G4
          END WHILE cap _ cap.canxt;
          GOTO step;
       (skipni): !SKIPN A2,0(A1); %instruction part for cary%
                                                              5S2B18
       (done):                                                5S2B19
       IF chflag THEN BEGIN
          IF (wnc _ wnc + chnc) > maxsl THEN
             dismes (2, $"New statement is too long --
             substitution not made.")
          ELSE
             BEGIN
             % copy rest of statement (past last match) %
                UNTIL chbp = sbp DO
                    ^wbp _ ^chbp;
             lit.L _ wnc;
             FIND SF(stid) ^sptr1 SE(stid) ^sptr2; %remove
             markers%
             cldsr($sptr1);
             ST stid _ *lit*;
             subcnt _ subcnt + chflag;
             END;
       END;
         %msftx();%
       RETURN;
       END.

%substitute support routines%
   (fltprc) % call a procedure through a filter %
   PROCEDURE (dent REF, dest REF, vs REF, procnam REF, param1,
   param2, param3, param4, param5);                           5S3A
      % Procedure description
         FUNCTION
            Given a structure, address to viewspecs, the address
            of the name of a procedure and up to five parameters,
            calls the procedure and works within the viewspecs
            specified if vs is non-zero.
            It is used by string manipulating commands over a
            filtered structure.
         ARGUMENTS
            dent REF, dest REF, vs REF, procnam REF, param1,
            param2, param3, param4, param5
         RESULTS
            none
         NON-STANDARD CONTROL
```

```
        calls the procedure in procnam
    GLOBALS
        none
    %
LOCAL last, vspec, save1, save2, savca, savus, da REF, sw
REF;
&dest _ ELEM #dest#[tppair];
&da _ lda();
curmkr _ dest;  curmkr[1] _ 1;
% save viewspecs %
    save1 _ da.davspec;
    save2 _ da.davspc2;
    savca _ da.dacacode;
    savus _ da.dausqcod;
    INVOKE(restrvs);
IF &vs THEN
    BEGIN
    da.davspec _ vs.vs1;
    da.davspc2 _ vs.vs2;
    da.dacacode _ vs.vscacode;
    da.dausqcod _ vs.vsusqcod;
    END
ELSE
    BEGIN
    da.davspec _ 0;
    da.davspec.vslev _ da.davspec.vstrnc _
        da.davspec.vsnamf _ da.davspec.vsdaft _
            da.davspec.vsindf _ -1;
    da.dacacode _ da.dausqcod _ 0;
    END;
CASE ELEM #dent#[cwtype] OF
    = 29: %- statement -%
        BEGIN
        dpset(dsprfmt, dest, endfil, dest);
        procnam(dest, param1, param2, param3, param4,
        param5);
        END;
    = 27 %- group -%, = 28 %- plex -%, = 26 %- branch -%:
        BEGIN
        dpset(dspallf, dest, endfil, endfil);
        dest _ grptst(dest, [&dest+d2sel]: last);
        vspec _ da.davspec;
        vspec.vsbrof _ vspec.vsplxf _ FALSE;
        %open sequence work area%
            &sw _ openseq(dest, last, vspec, da.davspc2,
            da.dausqcod, da.dacacode) ;
        % work on all statements in seq work area%
            WHILE seqgen(&sw) # endfil DO
                BEGIN
                IF inptrf THEN
                    BEGIN
                    dismes(1, $"user terminated process");
                    EXIT LOOP;
                    END;
                procnam(sw.swstid, param1, param2, param3,
                param4, param5);
```

```
                END;
            %close seq work area%
               closeseq(&sw);
            END;
         ENDCASE ABORT(badarg, $"command word number program
         error");
      % restore viewspecs %
         da.davspec _ save1;
         da.davspc2 _ save2;
         da.dacacode _ savca;
         da.dausqcod _ savus;
         DROP (restrvs);
      RETURN;
      (restrvs) CATCHPHRASE;                                    5S3A12
         CASE SIGNALTYPE OF
             = aborttype:
               BEGIN
               da.davspec _ save1;
               da.davspc2 _ save2;
               da.dacacode _ savca;
               da.dausqcod _ savus;
               CONTINUE;
               END;
            ENDCASE;
      END.

   (stbpget) PROCEDURE (stid, cnt); %statement byte pointer get%
                                                              5S3B
      LOCAL bp;
      swork _ stid; swork[1] _ cnt;
      fechc1(1, $swork);
      %swork[2] contains byte count+1, swork[4] has byte pointer%
      bp _ swork[4];
      IF bp .A 77B10 = 44B10 THEN
         bp _ bp - 430000000001B;
         %adjust char and word pos%
      RETURN (bp, swork[2]-1);
      END.

   (bkbp) PROCEDURE (bp); %back up byte pointer%              5S3C
      IF (bp _ bp + 07B10) < 0 THEN
         bp _ bp - 430000000001B;
      RETURN (bp);
      END.

   (sbdelck) PROCEDURE (ch, ttype); %check delimiter%         5S3D
      CASE ttype OF
         =numbrv: IF ch=D OR ch=". OR ch=", THEN RETURN(FALSE);
         =wordv: IF ch=LD THEN RETURN(FALSE);
         =visv: IF ch=PT THEN RETURN(FALSE);
         =invisv: IF ch=NP THEN RETURN(FALSE);
         ENDCASE;
      RETURN (TRUE);
      END.

   (subsinit) PROC( % initializes state for the substitute
```

```
command %                                                    5S3E
   % FORMAL ARGUMENTS %
      inittype,          %initialize (TRUE) or cleanup (FALSE)%
      textent,           % text entity type for the substitute %
      resultptr REF);    %addresses of allocated strings%
   LOCAL % VARIABLES %
      subdsp,  % ptr to substitute display buffer %
      ttype;   % text entity code %
   LOCAL STRING subm[30];  % string for "Subs = " message %
   REF subdsp, resultptr, textent;
   % ---------- %
   IF inittype THEN
      BEGIN
      % set ttype according to type of text entity-- This code
      should be removed after sbinit is changed to expect the
      new type of text entity codes %
         CASE textent OF
            = 2 %- character -%:        ttype _ charv;
            = 3 %- word -%:             ttype _ wordv;
            = 8 %- integer -%: ttype _ numbrv;
            = 4 %- visible -%: ttype _ visv;
            = 11 %- invisible -%:       ttype _ invisv;
            = 30 :      % -link- ttype _ linkv; %
               BEGIN
               dismes(1,$"Not Implemented yet...") ;
               subcnt _ 0;
               RETURN ;
               END;
            = 1 %- text -%:             ttype _ textv;
            ENDCASE ABORT (badarg, $"The type of the list of
            substitutions was bad.  Respecify the entire
            command.");
      % allocate a display buffer for the substitute processor
      %
         resultptr _ resultptr[1] _ 0; %will be used in
         cleanup. %
         resultptr _ &subdsp _ getstring( 640, $dspblk );
      % allocate an astr buffer for the substitute processor %
         resultptr[1] _ getstring( 500, $dspblk );
      % initialize the substitute state info %
         sbinit( $subhed, $subrec, &subdsp, resultptr[1],
         ttype );
      % initialize substitutions count %
         subcnt _ 0;
      END
   ELSE
      BEGIN
      % deallocate the display buffer if allocated %
         IF resultptr THEN freestring( resultptr:=0, $dspblk
         );
      % deallocate the astr if allocated %
         IF resultptr[1] THEN freestring( resultptr[1]:=0,
         $dspblk );
      % display the number of substitutions made %
         *subm* _ "Substitutions made: ", STRING(subcnt);
         dismes (2, $subm);
```

ABORT (badarg, $"Not implemented yet");

```
      END;
   RETURN; % TRUE return %
   END.

(sbinit) PROCEDURE(sbhed, sbrec, sbdsp, sbastr, sbttype);  5S3F
   % initialize substitute data structure %
   LOCAL j;
   POINTER sbhed;
   sbhed.sbrp _ sbrec;
   sbhed.sbdp _ sbdsp;
   sbhed.sbas _ sbastr;
   sbhed.sbtt _ sbttype;
   *[sbastr]* _ NULL;
   FOR j _ 0 UP UNTIL >=200B DO [sbdsp+j] _ 0;
   RETURN;
   END.

%   Display procs not used  -----

(sub1dsp) PROCEDURE( %% substitute prompting function %%   5S3I
   %% FORMAL ARGUMENTS %%
      textent, %% text entity type for the substitute %%
      newflag);        %% TRUE if new text entity is being
      collected %%
   LOCAL STRING string[50];    %% prompting string %%
   LOCAL TEXT POINTER tp1, tp2;
   LOCAL textptr REF;
   REF resultptr, textent, newflag;
   %% ------------ %%
   &textptr _ ELEM #textent#[tppair];
   IF nlmode = typewriter THEN crlf();
   IF newflag
      THEN *string* _ "New "
      ELSE *string* _ "for all occurrences of old ";
   tp1 _ textptr;  tp2 _ textptr[d2sel];
   *string* _ *string*, tp1 tp2;
   dismes (2, $string);
   RETURN;
   END.

(sub2dsp) PROCEDURE( %% substitute prompting function %%   5S3J
   %% FORMAL ARGUMENTS %%
      type,             %% type of selection made %%
      tptr);   %% ptr to string tptr record %%
   LOCAL tptr2, adstr[40];
   LOCAL STRING string[100];   %% prompting string %%
   REF resultptr, tptr, tptr2, type;
   %% ------------ %%
   CASE type OF
      = 8 %%- link -%%:
         BEGIN
         IF (tptr.stastr) AND
            ( NOT FIND SF(tptr) $(SP/TAB) ("(/"</"--") ) THEN
               ST tptr _ "(, SF(tptr) SE(tptr);
         IF (tptr.stastr) AND
            ( NOT FIND SE(tptr) $(SP/TAB) (")/">) ) THEN
```

```
                ST tptr _ SF(tptr) SE(tptr), ">;
            lnkprs( &tptr, $adstr);
            tptr _ adstr[ls];
               tptr[1] _ adstr[ls+1];
            [&tptr+d2sel] _ adstr[le];
               [&tptr+d2sel+1] _ adstr[le+1];
            END;
         ENDCASE;
      IF nlmode = fulldisplay
         AND NOT tptr.stastr THEN
            BEGIN
            &tptr2 _ &tptr + d2sel;
            *string* _ tptr tptr2;
            aplit( $string ); %% feedback bugged param %%
            END;
      RETURN;
      END.

   %
   (subpsave) PROCEDURE( % stashes away parameters for substitute
   %                                                          5S3L
      % FORMAL ARGUMENTS %
         newptr,  % ptr to the new entity %
         oldptr); % ptr to the old entity %
      LOCAL new2, old2;
      LOCAL STRING newstr[200], oldstr[200];
      REF resultptr, newptr, oldptr, new2, old2;
      % ------------ %
      % fetch parameters to local strings %
         &new2 _ &newptr + d2sel;
         &old2 _ &oldptr + d2sel;
         *newstr* _ newptr new2;
         *oldstr* _ oldptr old2;
      % check length of the old (target) string %
         IF oldstr.L = empty THEN
            ABORT(0, $"cannot substitute for a null text field"
            );
      % stash away the parameter strings %
         sbpush( $newstr, $oldstr, $subhed );
      RETURN($subhed);
      END.

   (sbpush) PROCEDURE(str1, str2, hed);                       5S3M
      %.....Documentation.....%
         %The test strings are sorted by initial character and
         chained together, with the head of the chain in a
         character-code indexed array subdsp.  This allows a very
         fast check whether a character can possibly begin a test
         string.  Each test-replacement pair is represented by a
         record (card) which is linked to others for the same
         initial test character through the canxt field.%
         %Global variables used:
            subcnt  count of substitutions
            lit     A-string for building up new statement
            swork   internal work area, see (stbpget)
         %
```

```
        % add pair to substitute list %
        LOCAL astr, len, len1, len2, subrp, asa, cap;
        POINTER hed, subrp, cap;
        REF str1, str2, astr;
        &astr _ hed.sbas;
        len _ astr.L;
        asa _ &astr + 1; %origin of text for A-string%
        len1 _ str1.L;
        len2 _ str2.L;
        *astr* _ *astr*, *str2*;
        IF hed.sbtt = numbrv THEN
           % pad replacement if shorter %
           FOR len1 UP UNTIL >=len2 DO
              *astr* _ *astr*, SP;
        *astr* _ *astr*, *str1*;
        subrp _ hed.sbrp;
        subrp.carnc _ len1;
        subrp.catnc _ len2;
        subrp.carbp _ conbp(asa,len+len2);
        subrp.catbp _ conbp(asa,len);
        subrp.canxt _ 0;
        %link card into appropriate chain%
           cap _ hed.sbdp + *str2*[1];
           IF [cap] = 0 THEN %empty chain%
              [cap] _ subrp
           ELSE BEGIN %link on end%
              cap _ [cap];
              WHILE cap.canxt DO cap _ cap.canxt;
              cap.canxt _ subrp;
           END;
        hed.sbrp _ subrp + lcard;
        RETURN;
        END.

     DECLARE bpary = (010677777777B, 3507B8, 2607B8, 1707B8,
     1007B8);
     (conbp) PROCEDURE(base,cn);                                    5S30
        % construct byte pointer assuming 5 characters per word.
        cn is the character number %
        LOCAL q, r;
        DIV cn / 5, q, r;
        RETURN(base+bpary[r]+q);
        END.

%terminate%
   (xterm)  %WM termination procedure%                              5T1
      PROCEDURE ;
         termmiddle();  %middle-end knows how to terminate%
         RETURN;
         END.
%transpose%
   %transpose PCP interface routine%
      (xtranspose) %Execute Transpose Command%                      5U1A
         PROCEDURE (
            %FORMALS%
               sourceptr REF,        %source pointer%
```

```
        destptr REF,            %destination pointer%
        vs REF,                 %viewspec string%
        txtrtn,         %if TRUE, return text of statement%
        tprtn           %if TRUE, return text pointer(s)%
        );
    LOCAL adstr[40];  % block for parsing links %
    LOCAL intrastmt, retry REF, source REF, destination REF;
    LOCAL STRING badmsg[200];
    LOCAL LIST tplist [8], txtlist[1];
%----------------------%
IF &sourceptr THEN &source _ ELEM #sourceptr#[tppair];
IF &destptr THEN &destination _ ELEM #destptr#[tppair];
#tplist# _;
#txtlist# _ ;
intrastmt _ FALSE;
CASE ELEM #sourceptr#[cwtype] OF
    = 30 %- link -%:
        BEGIN
        lnkprs( &source, $adstr);
        source _ adstr[ls];
            source[1] _ adstr[ls+1];
        [&source+d2sel] _ adstr[le];
            [&source+d2sel+1] _ adstr[le+1];
        END;
    ENDCASE;
CASE ELEM #destptr#[cwtype] OF
    = 30 %- link -%:
        BEGIN
        lnkprs( &destination, $adstr);
        destination _ adstr[ls];
            destination[1] _ adstr[ls+1];
        [&destination+d2sel] _ adstr[le];
            [&destination+d2sel+1] _ adstr[le+1];
        END;
    ENDCASE;
CASE ELEM #sourceptr#[cwtype] OF
    = 2 %- character -%, = 11 %- invisible -%, = 1 %- text
    -%, = 3 %- word -%, = 4 %- visible -%, = 8 %- integer
    -%, = 30 %- link -%:
        BEGIN
        IF source[1] = destination[1] THEN intrastmt _ TRUE;
        IF NOT nortnrings THEN
            clist(ctcmk, destination.stfile, source.stfile);
        IF NOT nodisplay THEN
            dpset(dsprfmt, destination, source, endfil);
        curmkr _ source;  curmkr[1] _ source[1];
        ctratex(&destination, &destination+d2sel, &source,
        &source+d2sel);
        IF NOT nortnrings THEN clupdt();
        %
        IF txtrtn THEN #txtlist# _ curmkr;
        IF tprtn THEN
            IF ELEM #sourceptr#[cwtype] = 2 THEN
                #tplist# _ destination, destination[1], source,
                source[1]
            ELSE
```

```
                #tplist# _ destination, destination[1],
                destination[d2sel], destination[d2sel+1],
                source, source[1], source[d2sel],
                source[d2sel+1];
            %
            END;
        = 29 %- statement -%:
            BEGIN
            IF NOT nortnrings THEN
               clist(ctcsp, destination.stfile, source.stfile);
            IF NOT nodisplay THEN
               dpset(dspstrc, destination, source, endfil);
            curmkr _ destination;  curmkr[1] _ 1;
            ctrasta(destination, source, &vs);
            IF NOT nortnrings THEN clupdt();
            %
            IF tprtn THEN
               #tplist# _ destination, 1, source, 1;
            %
            END;
        = 27 %- group -%, = 28 %- plex -%, = 26 %- branch -%:
            BEGIN
            IF NOT nortnrings THEN
               clist(ctcsp, destination.stfile, source.stfile);
            IF NOT nodisplay THEN
               dpset(dspstrc,destination,source,endfil);
            curmkr _ source;  curmkr[1] _ 1;
            ctragro(destination, destination[d2sel], source,
            source[d2sel], &vs);
            IF NOT nortnrings THEN clupdt();
            %
            IF tprtn THEN
               IF ELEM #sourceptr#[cwtype] = 26 THEN
                  #tplist# _ destination, 1, source, 1
               ELSE
                  #tplist# _ destination, 1, destination[d2sel],
                  1, source, 1, source[d2sel], 1;
            %
            END;
        ENDCASE
        IF NOT donthelp THEN
            BEGIN
            *badmsg* _ "The selection type was incorrect.  The
            allowed values are Branch, Character, ";
            *badmsg* _ *badmsg*, "Group, Integer, Invisible,
            Link, Plex, Statement, Text, Visible, or Word.";
            &retry _ HELP (badarg, $badmsg, cindex);
            source _ ELEM #retry#[tppair];
            REPEAT CASE (ELEM #retry#[cwtype]);
            END
        ELSE ABORT (badarg, $"The selection type was
        incorrect.");
    %
    IF tprtn THEN
        IF (txtrtn AND intrastmt) THEN RETURN(txtlist, tplist)
        ELSE RETURN (empty, tplist)
```

```
        ELSE
           IF (txtrtn AND intrastmt) THEN RETURN(txtlist, empty)
           ELSE RETURN;
        %
        RETURN;
        END.

  %transpose core routine%
     (ctragro)                %Core NLS Transpose Group Command%  5U2A
        %This routine transposes the groups defined by FIRST1 and
        FIRST2 (first group) and SECND1 and SECND2 (second group).
        The first group need not be ahead of the second in the file
        structure, but the routine assumes that FIRST1,2 and
        SECND1,2 define legal groups.  It checks to see that the
        groups don't overlap.
           If one group follows the other immediately, then the
           second group is merely deleted from the structure and
           inserted after the pred (or source) of the first; the
           first the is in the right position automatically.
           Otherwise, the routine finds the pred (or source) of
           each group.
              If the two groups are in the same file, it calls
              REMGRP to delete one group from the structure, then
              INSGRP to insert the group after the pred (or source)
              of the other group.  The same delete/insert sequence
              is followed for the other group.  Otherwise, it uses
              COPGROUP twice and XDELE twice to perform the
              requisite operations.%
        PROCEDURE (first1, first2, secnd1, secnd2, vsptr);
        REF vsptr;
        LOCAL
           work,     %temp%
           f1, f2, s1, s2,  %temps for return stids%
           gp1tgt, %target for group delimited by first1,first2%
           gp1dir, %direction for insertion of above group%
           gp2tgt, %target for group delimited by secnd1,secnd2%
           gp2dir; %direction for insertion of above group%
         %msnst();%
        first1 _ grptst(first1, first2 : first2);
        secnd1 _ grptst(secnd1, secnd2 : secnd2);
        trntst(first1, first2, secnd1, secnd2);
        IF getfhd(secnd1) THEN
           BEGIN
           gp1tgt _ getup(secnd2);
           gp1dir _ levdown;
           END
        ELSE
           BEGIN
           gp1dir _ levsuc;
           IF (gp1tgt _ getprd(secnd1)) = first2 THEN
              IF NOT &vsptr THEN
                 BEGIN %SECND1,2 follows FIRST1,2 directly%
                 IF NOT remgrp(first1, first2) THEN
                    ABORT(badtranspose,$"Illegal transpose");
                 insgrp(secnd2, levsuc, first1, first2);
                 RETURN;
```

```
            END;
         END;
      IF getfhd(first1) THEN
         BEGIN
         gp2tgt _ getup(first2);
         gp2dir _ levdown;
         END
      ELSE
         BEGIN
         gp2dir _ levsuc;
         IF (gp2tgt _ getprd(first1)) = secnd2 THEN
            IF NOT &vsptr THEN
               BEGIN %FIRST1,2 follows SECND1,2 directly%
               IF NOT remgrp(secnd1, secnd2) THEN
                  ABORT(badtranspose,$"Illegal transpose");
               insgrp(first2, gp2dir, secnd1, secnd2);
               RETURN;
               END;
         END;
      IF first1.stfile = secnd1.stfile THEN
         IF NOT &vsptr THEN
            BEGIN %intrafile transpose%
            IF NOT remgrp(secnd1, secnd2) THEN
               ABORT(badtranspose,$"Illegal transpose");
            insgrp(gp2tgt, gp2dir, secnd1, secnd2);
            IF NOT remgrp(first1, first2) THEN
               ABORT(badtranspose,$"Illegal transpose");
            insgrp(gp1tgt, gp1dir, first1, first2);
            END
         ELSE
            BEGIN %intrafile filtered transpose%
            IF gp1tgt = first2 THEN
               BEGIN
               IF getfhd(first1) THEN
                  BEGIN
                  gp1tgt _ getup(first1);
                  gp1dir _ levdown;
                  END
               ELSE
                  BEGIN
                  gp1dir _ levsuc;
                  gp1tgt _ getprd(first1);
                  END;
               END;
            mvdlfgrp(
               first1, first2, &vsptr, IF (work _
               getnxt(getend(first2))) # endfil THEN work ELSE
               getbck(first1), trnsflag, gp1tgt, gp1dir);
            IF (gp2tgt = first2) OR (gp2tgt = secnd2) THEN
               BEGIN
               IF getfhd(secnd1) THEN
                  BEGIN
                  gp2tgt _ getup(secnd1);
                  gp2dir _ levdown;
                  END
               ELSE
```

```
                BEGIN
                gp2dir _ levsuc;
                gp2tgt _ getprd(secnd1);
                END;
             END;
          mvdlfgrp(
             secnd1, secnd2, &vsptr, IF (work _
             getnxt(getend(secnd2))) # endfil THEN work ELSE
             getbck(secnd1), trnsflag, gp2tgt, gp2dir);
          END
       ELSE
          BEGIN %cross file transpose%
          IF NOT &vsptr THEN
             BEGIN
             f1 _ copgrp(gp1tgt, gp1dir, first1, first2, TRUE:
             f2);
             s1 _ copgrp(gp2tgt, gp2dir, secnd1, secnd2, TRUE:
             s2);
             END
          ELSE
             BEGIN
             f1 _ copfgrp(gp1tgt, gp1dir, first1, first2, &vsptr:
             f2);
             s1 _ copfgrp(gp2tgt, gp2dir, secnd1, secnd2, &vsptr:
             s2);
             END;
          cdelgro(first1, first2, 0);
          cdelgro(secnd1, secnd2, 0);
          secnd1 _ s1;  secnd2 _ s2;
          first1 _ f1;  first2 _ f2;
          END;
        %msfst();%
       RETURN(secnd1, secnd2, first1, first2);
       END.

(ctrasta)              %Core NLS Transpose Statement Command%
                                                          5U28
       PROCEDURE (stid1, stid2, vsptr);
       REF vsptr;
       LOCAL
          % variables for intra-file transpose %
          rngloc1, % ring location of stid1 %
          rngloc2, % ring location of stid2 %
          sub1, % psids of sub for stid1 %
          suc1, % psids of suc for stid1 %
          hedf1,
          tailf1,
          up1, % stid (!) of up of stid1 %
          pred1, % stid (!) of pred of stid1 %
          tailsub1, % stid of tail of sub of 1; zero if 1 has no
          sub %
          sub2, % psids of sub for stid2 %
          suc2, % psids of suc for stid2 %
          hedf2,
          tailf2,
          up2, % stid (!) of up of stid2 %
```

```
        pred2, % stid (!) of pred of stid2 %
        tailsub2, % stid of tail of sub of 2; zero if 2 has no
        sub %
        % variables for inter-file transpose %
        newsd1, %new stid, replaces stid1 in cross-file
        trpose%
        newsd2, %new stid, replaces stid2 in cross-file
        trpose%
        tmpstd; %temporary to save stid%
      LOCAL TEXT POINTER t1, t2;
        %msnst();%
      IF NOT &vsptr THEN
        BEGIN
        FIND SF(stid1) ^t1 SE(stid1) ^t2;
        cldtxt($t1, $t2);
        FIND SF(stid2) ^t1 SE(stid2) ^t2;
        cldtxt($t1, $t2);
        IF stid1.stfile = stid2.stfile THEN
          BEGIN  %intrafile transpose%
          IF stid1.stpsid = origin OR stid2.stpsid = origin
          THEN
            ABORT(badtranspose,$"Illegal transpose");
          % Save stid1 ring values and up and predecessor %
            lodent( stid1, rngtyp : rngloc1 );
            IF [rngloc1].rsid = 0 THEN ABORT(badstid, $"Bad
            statement identifier");
            sub1 _ [rngloc1].rsub;
            suc1 _ [rngloc1].rsuc;
            hedf1 _ [rngloc1].rhf;
            tailf1 _ [rngloc1].rtf;
            up1 _ getup(stid1);
            pred1 _ getprd(stid1);
            tailsub1 _ IF sub1 = stid1.stpsid THEN 0 ELSE
            getail( getsub(stid1));
          % Save stid2 ring values and up and predecessor %
            lodent( stid2, rngtyp : rngloc2);
            IF [rngloc2].rsid = 0 THEN ABORT(badstid,$"Bad
            statement identifier");
            sub2 _ [rngloc2].rsub;
            suc2 _ [rngloc2].rsuc;
            hedf2 _ [rngloc2].rhf;
            tailf2 _ [rngloc2].rtf;
            up2 _ getup(stid2);
            pred2 _ getprd(stid2);
            tailsub2 _ IF sub2 = stid2.stpsid THEN 0 ELSE
            getail( getsub(stid2));
          % Store stid1 ring values in stid2 ring (rngloc is
          still address of stid2 ring) %
            lodent( stid2, rngtyp : rngloc2 ); % Block may
            have gone out %
            IF [rngloc2].rsid = 0 THEN ABORT(badstid,$"Bad
            statement identifier");
            [rngloc2].rsub _  CASE sub1 OF
              = stid1.stpsid: stid2.stpsid;
              = stid2.stpsid: stid1.stpsid;
              ENDCASE sub1;
```

```
            [rngloc2].rsuc _ IF suc1 = stid2.stpsid THEN
            stid1.stpsid ELSE suc1;
            [rngloc2].rhf _ hedf1;
            [rngloc2].rtf _ tailf1;
            % Check head flag of stid1; fill in ring fields of
            up or predecessor %
               IF hedf1 THEN
                  BEGIN
                  IF up1 # stid2 THEN stosub( up1, stid2);
                  %stid1 was at head%
                  END
               ELSE
                  BEGIN
                  IF pred1 # stid2 THEN stosuc( pred1, stid2
                  );
                  END;
            % Make successor field of tail of substructure of
            1 (if any) point to 2 %
               IF tailsub1 # 0 AND tailsub1 # stid2 THEN
                  stosuc( tailsub1, stid2 );
         % Store stid2 ring values in stid1 ring %
            lodent( stid1, rngtyp : rngloc1); % Block may have
            gone out %
            IF [rngloc1].rsid = 0 THEN ABORT(badstid,$"Bad
            statement identifier");
            [rngloc1].rsub _ CASE sub2 OF
               = stid1.stpsid: stid2.stpsid;
               = stid2.stpsid: stid1.stpsid;
               ENDCASE sub2;
            [rngloc1].rsuc _ IF suc2 = stid1.stpsid THEN
            stid2.stpsid ELSE suc2;
            [rngloc1].rhf _ hedf2;
            [rngloc1].rtf _ tailf2;
            % Check head flag of stid2; fill in ring fields of
            up or predecessor %
               IF hedf2 THEN
                  BEGIN
                  IF up2 # stid1 THEN stosub( up2, stid1);
                  %stid2 was at head%
                  END
               ELSE
                  BEGIN
                  IF pred2 # stid1 THEN stosuc( pred2, stid1
                  );
                  END;
            % Make successor field of tail of substructure of
            2 (if any) point to 1 %
               IF tailsub2 # 0 AND tailsub2 # stid1 THEN
                  stosuc( tailsub2, stid1 );
         %Set up return values%
            newsd1 _ stid2;  newsd2 _ stid1;
         END
      ELSE
         BEGIN %cross file transpose%
         newsd1 _ ccopsta(stid1, levsuc, stid2, 0);
         upctbl(stid2, newsd1, stid2);
```

```
            newsd2 _ ccopsta(stid2, levsuc, stid1, 0);
            upctbl(stid1, newsd2, stid1);
            IF (tmpstd _ getsub(stid1)) # stid1 THEN
               cmovgro(newsd1, levdown, gethed(tmpstd),
               getail(tmpstd), 0);
            cdelgro(stid1, stid1, 0);
            IF (tmpstd _ getsub(stid2)) # stid2 THEN
               cmovgro(newsd2, levdown, gethed(tmpstd),
               getail(tmpstd), 0);
            cdelgro(stid2, stid2, 0);
            END;
         END
      ELSE
         BEGIN
         IF getsub(stid1) # stid1 THEN
         ABORT(badtranspose,$"Illegal transpose");
         IF getsub(stid2) # stid2 THEN
         ABORT(badtranspose,$"Illegal transpose");
         newsd1 _ ctragro(stid1, stid1, stid2, stid2, &vsptr:
         up1%garbage%, newsd2, up2%garbage%);
         END;
        %msfst();%
      RETURN(newsd1, newsd2);
      END.

   (trntst)                                                    5U2C
      %This routine checks that the groups specified by the
      transpose commands have legal bounds.  It does this by
      calling MOVTST to see if FIRST1,2 are in the group bounded
      by SECND1,2 or SECND1,2 in the group bounded by FIRST1,2.%
      %-------------%
      PROCEDURE(first1, first2, secnd1, secnd2);
      movtst(first1, secnd1, secnd2);
      movtst(first2, secnd1, secnd2);
      movtst(secnd1, first1, first2);
      movtst(secnd2, first1, first2);
      RETURN;
      END.

   (ctratex) PROCEDURE(bug1, bug2, bug3, bug4)              %Core
   NLS Transpose Text Command% ;                              5U2D
      REF bug1, bug2, bug3, bug4;
      (tempstr) STRING [20000];                               5U2D2
        %msntx();%
      cldtxt(&bug1, &bug2);
      cldtxt(&bug3, &bug4);
      IF POS SF(bug1) = SF(bug3) THEN
         IF POS bug1 = bug3 AND POS bug2 = bug4 THEN RETURN ELSE
         IF POS bug1 < bug3 THEN
            ST bug1 _ SF(bug1) bug1, bug3 bug4, bug2 bug3,
               bug1 bug2, bug4 SF(bug1)
         ELSE
            ST bug1 _ SF(bug1) bug3, bug1 bug2, bug4 bug1,
               bug3 bug4, bug2 SF(bug1)
      ELSE
         BEGIN
```

```
            *tempstr* _ bug1 bug2;
            ST bug1 _ SF(bug1) bug1, bug3 bug4, bug2 SE(bug1);
            ST bug3 _ SF(bug3) bug3, *tempstr*, bug4 SE(bug3);
            END;
          %msftx();%
        RETURN;
        END.

%trim%
   %trim PCP interface routine%
      (xtrim) %Execute Trim Command%                                    5V1A
         PROCEDURE
            %FORMALS%
               (numvers);    %number of versions%                     5V1A1A1
         LOCAL tlngth; % temporary for string length %
         %----------------------%
         *lit* _ "Trimmed Files Are:", CR, LF;
         tlngth _ lit.L;
         ctridir(numvers, $lit);
            %trim connected directory%
         IF ( lit.L > tlngth ) THEN
            typelit(1, $lit)
         ELSE typelit(1, $"No Files Trimmed" );
         RETURN;
         END.

   %trim core routines%
      (ctridir)   % UB: core NLS Trim connected Directory procedure
      %
      PROCEDURE (nver, astr % => no value %);                           5V2A
         % Procedure description
            FUNCTION
               none
            ARGUMENTS
               nver - INTEGER - number of versions of each file to
               keep
               astr - STRING - string to receive results of trim
            RESULTS
               none
            NON-STANDARD CONTROL
               none
            GLOBALS
               none
            EXAMPLE
               none
            %
         % Declarations %
            LOCAL
               fjfn, % main jfn %
               jfnflgs,      % left half flags for a group jfn %
               pcjfn,        % jfn of partial copy %
               flags,        % bits indicating * typed for file name
               fields %
               ntver;        % actual number of versions deleted %
            LOCAL STRING
               jfnname[200], % complete name from jfns %
```

```
        errstring[200],       % eror message string %
        pstring[200]; % temporary string %
    REF astr;
% initial variables %
    fjfn _ jfnflgs _ pcjfn _ flags _ 0;
% must keep 1 or more versions %
    IF nver <= 0 THEN err( $"must keep 1 or more versions"
    );
% set up flags indicating file name of *.*;* %
    flags.fstar _ flags.estar _ flags.vstar _ TRUE;
% get main jfn (old file only, group jfn) %
    IF NOT (fjfn _ sgtjfn( gtjoif .V gtjstr + 0, $dirsfname,
    $errstring : jfnflgs ) )THEN err( $errstring );
%set jfn flags to get highest version for TOPS20%
    IF tops20flag THEN
        jfnflgs _ (jfnflgs .V 400000B) .A 777777B;
% now loop to get all files in the group %
    LOOP
        BEGIN
        % check to make sure this not a PC with * for
        extension name %
            IF NOT chkpcs( fjfn, flags) THEN GOTO gnxtjfn;
        % get actual file name into string for use later
        (maybe) %
            R3 _ 011100B6 + 1; % <directory>file.ext %
            jfnflink( fjfn, $jfnname, R3);
        % if this is same as previous one then go to next one
        %
            IF *jfnname* = *pstring* THEN GOTO gnxtjfn;
            *pstring* _ *jfnname*;
        % get PC jfn and find out if this user can access
        this PC %
            IF NOT getpcjfn( fjfn, $pcjfn, $errstring, FALSE)
            THEN
                BEGIN   % this user can't access this file %
                *astr* _ *astr*, "   ***   ", *jfnname*, " not
                trimmed because [", *errstring*, "]", CR, LF;
                GOTO gnxtjfn;
                END;
        % now delete the file (and its partial copy) %
            IF NOT triflandpc( fjfn, pcjfn, nver, $errstring :
            ntver) THEN
                BEGIN   % can't delete file (or partial copy) %
                *astr* _ *astr*, "   ***   ", *jfnname*, " not
                trimmed because ", *errstring*, CR, LF;
                GOTO gnxtjfn;
                END;
        % now tell the user if we have trimmed any files %
            IF ntver > 0 THEN
                BEGIN
                *astr* _ *astr*, "   ", *jfnname*, "  ",
                STRING( ntver ), " file";
                IF ntver > 1 THEN *astr* _ *astr*, "s;
                IF pcjfn THEN
                    IF ntver > 1 THEN *astr* _ *astr*, " (and
                    their partial copies)"
```

```
                ELSE *astr* _ *astr*, " (and it's partial
                copy)";
              *astr* _ *astr*, " deleted.", CR, LF;
              END;
        % now get the next file in the group %
          (gnxtjfn):                                        5V2A8A8A
          % now get rid of the jfn for the partial %
            IF NOT SKIP !rljfn( pcjfn ) THEN NULL;
          R1.LH _ jfnflgs;
          R1.RH _ fjfn;
          IF NOT SKIP !gnjfn( R1 ) THEN EXIT LOOP;
        END;
    % now get rid of any lingering jfns %
      IF NOT SKIP !rljfn( fjfn ) THEN NULL;
      IF NOT SKIP !rljfn( pcjfn ) THEN NULL;
    % all done now, so return %
     RETURN;
    END.   %%
```

```
%undelete%
   %undelete PCP interface routine%
      (xundelete) %Execute Undelete Command%                          5W1A
         PROCEDURE
            %FORMALS%
               (entity REF,  %entity type%
               filptr REF);  %filename pointer%
            LOCAL stid, tlngth, filename REF, rhostn;
            LOCAL STRING filestr[200];
            %----------------------%
            &filename _ ELEM #filptr# [tppair];
            CASE ELEM #entity# [cwtype] OF
               = 51 %- file -%:
                  BEGIN
                  *lit* _ "Undeleted Files Are:", CR, LF;
                  tlngth _ lit.L;
                  rhostn _ lnbfls( &filename, 0, $filestr);
                  cundfil(rhostn, $filestr, $lit);
                  IF ( lit.L <= tlngth ) THEN
                     *lit* _ "No Files Undeleted";
                  typelit(1, $lit);
                  END;
               = 61 %- archive -%: %file%
                  %
                  cundarcfil(&filename, &filename+d2sel);
                  %
               = 53 %- modifications -%: %to file%
                  BEGIN
                  stid _ orgstid;
                  stid.stfile _ lcfile();
                  clist (ctlcfm, stid.stfile, nofile);
                  dpset(dspallf, stid, endfil, endfil);
                  cundmodfil(stid.stfile);
                  <IOEXEC, unlkclist> (); %check clist items%
                  clupdt ();
                  END;
               ENDCASE err(notyet);
            RETURN;
            END.

   %undelete core routines%
      (cundfil)   % GB: undelete a group of files and partial copies
      %
      PROCEDURE (rhstn, fname, astr % => no value %);                 5W2A
         % Procedure description
            FUNCTION
               Undeletes the group of files specified by 'fname' on
               host 'rhostn'.  The files should have previously been
               deleted, but no error will occur if they had not
               been.
            ARGUMENTS
               rhstn - INTEGER -
                  number of the host on which the files reside.
                  Only the local host ('lhostn') is currently
                  implemented.
```

```
        fname - STRING -
           file group name string in TENEX format.  Connected
           directory is assumed if a directory is not
           specified.  Examples:
              "*.*"
              "<SMITH>*.PC"
              "<SMITH>FOO.NLS;3"
        astr - STRING - string to receive list of undeleted
        files
     RESULTS
        none
     NON-STANDARD CONTROL
        Error if
           host is not the local host
           file doesn't exist
     GLOBALS
        lhostn, gtjoif, gtjstr, gtjidl - read only
     EXAMPLE
        cundfil(lhostn, $"FOO.*", $string)
     %
  % Declarations %
     LOCAL
        fjfn, % main jfn %
        jfnflgs,      % left half flags for a group jfn %
        pcjfn,        % jfn of partial copy %
        flags;        % bits indicating * typed for file name
        fields %
     LOCAL STRING
        uname[200],           % complete name as entered by
        user %
        udirname[40], % user input directory name %
        ufilname[40], % user input file name %
        uextname[40], % user input extension name %
        uverno[40],           % user input version number %
        ulftovr[40],  % user input beyond version number %
        jfnname[200], % complete name from jfns %
        errstring[200],       % eror message string %
        tstring[200]; % temporary string %
     REF fname, astr;
  % initial variables %
     fjfn _ jfnflgs _ pcjfn _ flags _ 0;
  % find out if remote or local (disk) file %
     CASE rhstn OF
        = lhostn:  NULL;
        ENDCASE err( $"remote file manipulations not
        implemented yet" );
  % now parse user input strings %
     parseinput( &fname, $udirname, $ufilname, $uextname,
     $uverno, $ulftovr, $uname, $flags);
  % get main jfn (old file only, ignore deleted, group jfn) %
     IF NOT (fjfn _ sgtjfn( gtjoif .V gtjstr .V gtjidl + 0,
     $uname, $errstring : jfnflgs ) )THEN err( $errstring );
  % now find out if we support this type of file %
     IF NOT chkdev( fjfn ) THEN err( $"only disk files
     supported" );
  % now loop to get all files in the group %
```

```
        LOOP
            BEGIN
            % if this file already undeleted, then go on to next
            one %
               !gtfdb( fjfn, 1B6 + 1B, $R3);
               IF NOT R3 .A 4B10 THEN GOTO gtnxtjfn;
            % check to make sure this not a PC with * for
            extension name %
               IF NOT chkpcs( fjfn, flags) THEN GOTO gtnxtjfn;
            % get actual file name into string for use later
            (maybe) %
               R3 _ 011110B6      %  directory file ext ver %
                  + (IF jfnflgs.lhjb9  THEN 1B6 ELSE 0) %
                  ;Protection %
                  + (IF jfnflgs.lhjb10 THEN 1B5 ELSE 0) %
                  ;Account %
                  + (IF jfnflgs.lhjb11 THEN 4B4 ELSE 0) % ;T %
                  + 1;              % with proper file punctuation
                  %
               jfnflink( fjfn, $jfnname, R3);
            % get PC jfn and find out if this user can access
            this PC %
               IF NOT getpcjfn( fjfn, $pcjfn, $errstring, FALSE)
               THEN
                  BEGIN   % this user can't access this file %
                  *tstring* _ *jfnname*, " not undeleted because
                  [", *errstring*, "];
                  IF flags THEN
                     BEGIN
                     *astr* _ *astr*, "   ***   ", *tstring*, CR,
                     LF;
                     GOTO gtnxtjfn;
                     END
                  ELSE
                     BEGIN
                     IF NOT SKIP !rljfn( fjfn) THEN NULL;
                     err( $tstring );
                     END;
                  END;
            % now undelete the file (and its partial copy) %
               IF NOT undflandpc( fjfn, pcjfn, $errstring) THEN
                  BEGIN   % can't undelete file (or partial copy)
                  %
                  *tstring* _ *jfnname*, " not undeleted because
                  ", *errstring*;
                  IF flags THEN
                     BEGIN
                     *astr* _ *astr*, "   ***   ", *tstring*, CR,
                     LF;
                     GOTO gtnxtjfn;
                     END
                  ELSE
                     BEGIN
                     IF NOT SKIP !rljfn( fjfn) THEN NULL;
                     IF NOT SKIP !rljfn( pcjfn) THEN NULL;
                     err( $tstring );
```

```
                END;
            END;
        % now tell the user we have undeleted this file %
          *astr* _ *astr*, "   ", *jfnname*;
          IF pcjfn THEN *astr* _ *astr*, " and its partial
          copy";
          *astr* _ *astr*, CR, LF;
        % now get rid of the jfn for the partial %
          IF NOT SKIP !rljfn( pcjfn ) THEN NULL;
        % now get the next file in the group %
          (gtnxtjfn):                                        5W2A8A9A
          R1.LH _ jfnflgs;
          R1.RH _ fjfn;
          IF NOT SKIP !gnjfn( R1 ) THEN EXIT LOOP;
        END;
    % now get rid of any lingering jfns %
      IF NOT SKIP !rljfn( fjfn ) THEN NULL;
      IF NOT SKIP !rljfn( pcjfn ) THEN NULL;
    % all done now, so return %
      RETURN;
    END.   %%
```

```
(cundmodfil)   % UB: core NLS Undelete Modifications to File
proc %
PROCEDURE (fileno % => no value %);                                5W2B
   % Procedure description
      FUNCTION
         Undeletes the Partial Copy associated with the NLS
         file 'fileno', thereby restoring the modifications
         contained in that partial copy.  Then updates the
         file header to replect the fact that the file is
         again locked (has a PC).
      ARGUMENTS
         fileno - INTEGER - NLS file number of the file
      RESULTS
         none
      NON-STANDARD CONTROL
         Error if the partial copy is not deleted or if
         undeleting it fails.
      GLOBALS
         none
      EXAMPLE
         cundmodfil(stid.stfile)
      %
   % Declarations %
      LOCAL fl;  REF fl;
      LOCAL STRING errstr[100];
      &fl _ flntadt(fileno);
      IF fl.flpart THEN err($"Partial copy is already
      undeleted.");
   writeout(fileno);
   IF NOT makepc(&fl, fileno, TRUE, $errstr) THEN
      err($errstr);
   RETURN;
   END.   %%
```

```
%update%
   %update PCP interface routine%
      (xupdate) %Execute Update Command%                              5X1A
         PROCEDURE (
            %FORMALS%
               entity REF,             %entity type%
               nnameptr REF,           %newnm pointer%
               window,                 %window number%
               rtnlist REF);           %return arg%
            LOCAL newnm REF, tp2 REF, fileno, stid, retry;
            LOCAL STRING filstring[200], newstring[200];
         %--------------------%
         IF &nnameptr THEN &newnm _ ELEM #nnameptr#[tppair];
         stid _ orgstid;
         stid.stfile _  fileno _ dspfile(window);
         IF NOT nodisplay THEN dpset(dsprfmt, stid, endfil, endfil);
         CASE ELEM #entity#[cwtype] OF
            = 52 %- old -%:
               cupdfil (fileno, oldversion, 0);
            = 51 %- new -%:
               cupdfil (fileno, newversion, 0);
            = 53 %- compact -%: %file%
               BEGIN
               IF NOT nodisplay THEN dpset(dspallf, stid, endfil,
               endfil);
               clist(15, fileno, 0);
               cupdfil (fileno, upcompact, 0);
               % clupdt();  exists in cupdfil %
               END;
            = 54 %- rename -%: %file%
               BEGIN
               % move file name to local string %
                  CASE lnbfls( &newnm, 0, $newstring) OF
                     = lhostn: NULL;
                     ENDCASE
                        ABORT(noremote, $"Remote File Manipulations
                        Not Implemented Yet");
               cupdfil (fileno, newname, $newstring);
               END;
            ENDCASE
            IF NOT donthelp THEN
               BEGIN
               retry _ HELP (badarg, $"The type argument was
               incorrect.  The allowed values are Old, New, Compact,
               or Rename.", cindex);
               REPEAT CASE (retry);
               END
            ELSE ABORT (badarg, $"The type argument was
            incorrect.");
         *filstring* _ NULL;
         %-NSW%                                                       5X1A9
            filnam( fileno, $filstring);
         %-NSW%                                                      5X1A10
         %+NSW%                                                      5X1A11
            rfilnam( fileno, $filstring);
```

```
        %+NSW%                                                    5X1A12
        % until get a SHOWSTATUS in CML
           #rtnlist#[1] _ *filstring*;
        %
        dismes(1, $filstring);
        RETURN;
        END.

   %update core routines%
      (cupdfil)              %Core NLS Update File command %      5X2A
         PROCEDURE (fileno, type, namstr);
         CASE type OF
            = oldversion, = newversion:
               IF NOT updtfl (fileno, type, 0) THEN
                  ABORT (nomod, $"Update unnecessary: no
                  modifications");
            = newname:
               IF NOT updtfl (fileno, newname, namstr) THEN
                  ABORT (nomod, $"Update unnecessary: no
                  modifications");
            = upcompact:
               compactfile (fileno);
            ENDCASE ABORT (badupdate, $"Illegal update request");
         RETURN;
         END.

      (updtfl)   PROCEDURE   % does Update File command %        5X2B
         (fileno, % NLS file number of file to be updated %
         vertype,  % newversion if Update new, oldversion if update
         old %
         filname  % Address of string containing new name of updated
            file.  May be zero if we want to use the current name
            which is available through the file status table. %
         );
         %Returns true if file updated, false if no changes, file
         not updated.%
         LOCAL
            fl, dt, fhd, stid, flags, rn, stdb, sdbblk, rngblk,
            pgindx, lokdir, lokinit, newjfn, fhdpag, count, temp;
         LOCAL STRING nfname[200], locstr[200], datestr[40];
         LOCAL TEXT POINTER ptr;
         REF fl, dt, rn, filname;
         % get address of file status table entry for this file %
            &fl _ flntadr(fileno);
         % get file link for the old file %
            jfnflink( fl.florig, $locstr, 011110B6+1);
         % used to call ckdiracc to check for write access%
         % if file in browse mode, put out error message, don't
         update. %
            IF fl.flbrws THEN
               BEGIN
               *lit* _ *[fl.flastr]*, " is in browse mode";
               ABORT (badupdate, $lit);
               END;
         IF fl.flexis AND fl.florig THEN
            IF fl.flpart OR (vertype = newversion AND &filname)  OR
```

```
        (vertype = newname AND &filname)THEN
           BEGIN
           IF vertype = newversion OR vertype = newname THEN
           %open new version with write access%
              BEGIN
              IF &filname THEN
                 BEGIN
                 IF NOT (newjfn _ lgetjfn
                    (0,&filname, $nlsext, gtjoof, $lit)) THEN
                       ABORT (badupdate, $lit);
                 jfnstr(newjfn, $nfname);
                 IF NOT sysopen(newjfn, readwrite, random, $lit)
                 THEN
                    BEGIN
                    reljfn(newjfn);
                    ABORT (badupdate, $lit);
                    END;
                 END
              ELSE
                 BEGIN
                 *nfname* _ *[fl.flastr]*;
                 count _ nfname.L;
                 WHILE *nfname*[count] NOT= fvrdchar DO
                    count _ count-1; %fvrdchar is ";" for tenex,
                    "." for tops20%
                 nfname.L _ count-1; %delete version number from
                 name%
                 flags _ getgtjflg(write, origff, dfltvrs);
                 IF NOT (newjfn _ sgtjfn(flags, $nfname, $lit))
                 THEN
                    err($lit)
                 ELSE IF NOT sysopen(newjfn, readwrite, random,
                 $lit) THEN
                    BEGIN
                    reljfn(newjfn);
                    ABORT (badupdate, $lit);
                    END;
                 END
              END
           ELSE %close and reopen old version with write access%
              BEGIN
              IF NOT writeout(fileno, $lit)
              OR NOT sysclose(fl.florig .V 4B11, $lit) THEN
                 ABORT (badupdate, $lit);
              IF NOT sysopen(fl.florig, readwrite, random, $lit)
              THEN
                 BEGIN
                 %failed to get write access, reopen with read
                 access%
                 dismes (2, $lit);
                 IF NOT rdhdr(fileno, $lit) THEN ABORT
                 (badupdate, $lit);
                 IF NOT sysopen(fl.florig, read, random, $lit)
                 THEN
                    ABORT (badupdate, $lit);
                 ABORT (statesig, 0);
```

```
            END;
         newjfn _ fl.florig;
         IF NOT rdhdr(fileno, $lit) THEN ABORT (badupdate,
         $lit);
         IF lit.L > empty THEN
            BEGIN
            dismes (2, $lit);
            END;
         END;
      % get partial copy header address %
         fhd _ filehead[fileno];
      % first do the ring blocks %
         &rn _ fhd - $filhed + $rngst;
         stid _ 0;
         stid.stfile _ fileno;
         rngblk _ 0;
         DO
            BEGIN
            IF rn.rfexis THEN
               BEGIN
               IF (rn.rfpart OR vertype = newversion OR
               vertype = newname) THEN
                  BEGIN
                  stid.stblk _ rngblk;
                  pgindx _ lodent(stid, rngtyp);
                  % copy page to the file %
                  updtpg(rngblk + rngbas, pgindx, newjfn);
                  rn.rfcore _ 0;
                  END;
               END;
            &rn _ &rn + 1;
            END
         UNTIL (rngblk _ rngblk + 1) = rngm;
      % now do the sdb blocks %
         stdb _ 0;
         stdb.stfile _ fileno;
         &dt _ fhd - $filhed + $dtbst;
         sdbblk _ 0;
         DO
            BEGIN
            IF dt.rfexis THEN
               BEGIN
               IF (dt.rfpart OR vertype = newversion OR
               vertype = newname) THEN
                  BEGIN
                  stdb.stblk _ sdbblk;
                  pgindx _ lodent(stdb, sdbtyp);
                  % map in the file page %
                  updtpg(sdbblk + dtbbas, pgindx, newjfn);
                  dt.rfcore _ 0;
                  END;
               END;
            &dt _ &dt + 1;
            END
         UNTIL (sdbblk _ sdbblk+1) = dtbm;
      % finally do the header %
```

```
% map in the new file header %
   R1.LH _ newjfn;
   R1.RH _ 0;
   R2.LH _ 485;
   R2.RH _ $ckpag / 1000B;
   R3 _ 14B10;
   !JSYS pmap;
% get addr of file header (including type info) %
   pgindx _ fl.flhead;
   fhdpag _ crpgad[pgindx];
% block transfer old file header to new %
   mvbfbf (fhdpag, $ckpag, 512);
%remove the old header%
   R1 _ -1;
   R2.LH _ 4B5;
   R2.RH _ fhdpag / 512;
   R3 _ 0;
   !JSYS pmap;
fhd _ $ckpag + fbhdl;
% update last write time %
   [fhd - $filhed + $lwtim] _ gtadcall();
% update last write identification %
   [fhd - $filhed + $finit] _ cinit;
%set partial copy bits in new header for
subsequent updating of origin statement%
   &rn _ fhd - $filhed + $rngst;
   rngblk _ 0;
   DO
      BEGIN
      IF rn.rfexis THEN
         BEGIN
         %pretend like pc for later edit of origin
         statement%
            rn.rfpart _ TRUE;
         END;
      &rn _ &rn + 1;
      END
   UNTIL (rngblk _ rngblk + 1) = rngm;
   &dt _ fhd - $filhed + $dtbst;
   sdbblk _ 0;
   DO
      BEGIN
      IF dt.rfexis THEN
         BEGIN
         %pretend like pc for later edit of origin
         statement%
            dt.rfpart _ TRUE;
         END;
      &dt _ &dt + 1;
      END
   UNTIL (sdbblk _ sdbblk+1) = dtbm;
% remove the page %
   R1 _ -1;
   R2.LH _ 4B5;
   R2.RH _ $ckpag / 1000B;
   R3 _ 0;
```

```
            !JSYS pmap;
        % open the original file with read access %
          IF (vertype = newversion OR vertype = newname)
          THEN
            BEGIN
            %Propogate  access%
            IF vertype = newversion THEN
              BEGIN
            %read old access%
              R1 _ fl.florig;
              R2 _ 1000024B;
              R3 _ $R3;
              !JSYS gtfdb;
            %propogate to new file%
              R1 _ 24B6 .V newjfn;
              R2 _ 0;
              R2.acctyp _ 36M;
              !JSYS chfdb;
              END;
            %propagate archive "don't archive" and "don't
            delete" bits%
              IF vertype # newname THEN
                BEGIN
                !gtfdb(fl.florig, 1000017B, $R3);
                !chfdb(17B6 .V newjfn, 11B10); %R3 set
                from gtfdb%
                END;
            IF NOT <IOEXEC, writeout>(fileno, $lit)
            OR NOT <IOEXEC, delpc>(fileno, $lit)
            OR NOT <IOEXEC, sysclose>(fl.florig, $lit) THEN
              ABORT (badupdate, $lit);
            fl.flpart _ newjfn; %to fake out write psi
            stuff, in updating origin, below%
            filepart[fileno] _ TRUE;
            END
          ELSE
            BEGIN
            IF NOT <IOEXEC, writeout>(fileno, $lit)
            OR NOT <IOEXEC, delpc>(fileno, $lit) THEN
              ABORT (badupdate, $lit);
            fl.flpart _ fl.florig; %to fake out write psi
            stuff, in updating origin, below%
            filepart[fileno] _ TRUE;
            END;
          IF NOT rdhdr(fileno, $lit) THEN ABORT (badupdate,
          $lit);
          % update the origin statement and filst %
            fl.florig _ 0;
            *nfname* _ NULL;
            %-NSW%                                    5X2B14A8D3
              jfnflink(fl.flpart,$nfname,01111086+1);
            %-NSW%                                    5X2B14A8D4
            %+NSW%                                    5X2B14A8D5
              rfilnam(fileno, $nfname);
            %+NSW%                                    5X2B14A8D6
            uplsfnm($locstr, $nfname);
```

```
            %update file names in link stack%
            *datestr* _ NULL;
            <AUXCOD, getdat>($datestr);
            ptr _ orgstid;
            ptr.stfile _ fileno;
            ptr[1] _ 1;
            setrot($ptr, $nfname, $datestr);
            *nfname* _ NULL;
            jfnstr( fl.flpart, $nfname);
            IF fl.flastr THEN freestring( fl.flastr := 0,
            $dspblk);
            fl.flastr _ getstring( nfname.L + 1, $dspblk);
            *[fl.flastr]* _ *nfname*;
            !JSYS gjinf;
            %-NSW%                                      5X2B14A8D20
               temp _ R1; %loggin dir%
            %-NSW%                                      5X2B14A8D21
            %+NSW%                                      5X2B14A8D22
               temp _ R2; %connected dir%
            %+NSW%                                      5X2B14A8D23
            IF fl.flpcst THEN freestring( fl.flpcst := 0,
            $dspblk);
            fl.flpcst _ cpcnam(fl.flastr, temp);
            fldirectory(&fl);
         % finish cleaning up %
            dumpc(fileno, FALSE); %turn off pc bits in
            file%
            IF NOT <IOEXEC, writeout>(fileno, $lit)
            OR NOT <IOEXEC, sysclose>(fl.flpart .V 4B11,
            $lit) THEN
               ABORT (badupdate, $lit);
            fl.florig _ fl.flpart := 0;
            filepart[fileno] _ FALSE;
            IF NOT sysopen(fl.florig, read, random, $lit)
            THEN
               ABORT (badupdate, $lit);
         %delete old versions%
            %-NSW%                                      5X2B14A8F1
               delovsrns(fl.florig, nvtk);
            %-NSW%                                      5X2B14A8F2
            %+NSW%                                      5X2B14A8F3
               %Retains a backup copy as filename.BKUP%
               wmupdate(fileno);
            %+NSW%                                      5X2B14A8F4
         IF NOT rdhdr(fileno, $lit) THEN ABORT (badupdate,
         $lit);
         IF lit.L > empty THEN
            BEGIN
            dismes (2, $lit);
            END;
      END
   ELSE %check to see if it is locked without a partial
   copy%
      %

      There is some danger in this.  If the file is being
```

```
            accessed by the same user at another console we are
            going to unlock the file out from under him.

            %
            BEGIN
            R1 _ fl.florig;
            R2 _ 1000024B;
            R3 _ $R3;
            !JSYS gtfdb;
            lokdir _ R3.lkdirn;
            lokinit _ R3.lkinit;
            !JSYS gjinf;
            IF R1 = lokdir AND lokinit = cinit THEN
               BEGIN  %file is locked by this user...unlock it%
               IF NOT maywrt(fileno) THEN err($"No write
               access");
               R1.LH _ 24B;
               R1.RH _ fl.florig;
               R2 _ 0;
               R2.lkinit _ R2.lkdirn _ 36M;
               R3 _ 0;
               !JSYS chfdb;
               END
            ELSE
               RETURN(FALSE); %Update unnecessary%
            END;
      RETURN(TRUE); %File has been updated%
      END.

(updtpg) PROCEDURE % used in update to map a page from the
destination file into the update workarea page (beginning at
ckpag), then block transfers a page from the original file or
the PC to the workarea; finally puts this page back into the
destination file page by pmapping it back out (TENEX page
tables keep track of this); page removed from NLS core page
status table %                                              5X2C
      (fpage,  %  Destination page in the file being constructed
      which
         we are working on %
      pgindx,  % Index into core page status table of page in
      file
         (original or partial copy) to be sent to the destination
         file %
      newjfn   % jfn of the file being updated to %
      );
      LOCAL blk, ct;
      REF ct;
      % map in the destination file page%
         R1.LH _ newjfn;
         R1.RH _ fpage;
         R2.LH _ 4B5;
         R2.RH _ $ckpag / 1000B;
         R3 _ 14B10;
         !JSYS pmap;
      % get the address of the old file page (which is in core)
      to be sent to the destination %
```

```
        &ct _ $corpst + pgindx;
        blk _ crpgad[pgindx];
    % block transfer the from the old location to the workarea
    %
        mvbfbf (blk, $ckpag, 512);
    %remove sent page from core-- doing this sends it to the
    destination file%
        R1 _ -1;
        R2.LH _ 485;
        R2.RH _ blk / 512;
        R3 _ 0;
        !JSYS pmap;
    % Remove all traces of it from the NLS core page status
    table %
        ct _ 0;
    RETURN;
    END.


%used to build statements%
    (compactfile)  PROCEDURE    % does Compacted update File
    command %                                                    5X2D1
        (ofn);      % old NLS file number (of the file being
        output) %                                                5X2D1A
        %-----------------%
        LOCAL njfn, nfn, nfl, ofl, temp, oprvsts, curskpachk;
        LOCAL TEXT POINTER tp;
        LOCAL STRING nfilnam[200], locstr[200];
        REF nfl, ofl;
        curskpachk _ skpachk;
        INVOKE (sigassign);
        skpachk _ TRUE;
        &ofl _ flntadr(ofn);
        oprvsts _ rdprvsts (ofn);
        %used to call ckdiracc to check write access%
        *nfilnam* _ *[ofl.flastr]*;
        incrvrsn($nfilnam);
        IF NOT (njfn _ lgetjfn (0, $nfilnam, $nlsext, gtjoof,
        $lit)) THEN
            ABORT (ofilerr, $lit);
        jfnstr (njfn, $nfilnam);    % get full file name %
        nfn _ addfil (njfn, $nfilnam);
        %make it a partial copy%
            &nfl _ flntadr (nfn);
            nfl.flpart _ nfl.florig;
            filepart[nfn] _ TRUE;
        IF NOT sysopen (nfl.flpart, readwrite, random, $lit)
        THEN
            ABORT (ofilerr, $lit);
        clsid($clhead); %convert psid's to sid's in
        correspondence list%
        copfil (ofn, nfn, $clhead);
        %update the new origin statement%
            *lit* _ NULL;
            <AUXCOD, dtfrmt> ([filhdr (nfn) + $lwtim - $filhed],
            $lit);
```

```
        tp _ 0;
        tp.stfile _ nfn;
        tp.stpsid _ origin;
        tp[1] _ 1;
        % get file name in link form %
           jfnflink( nfl.florig, $locstr, 011110B6+1);
        setrot ($tp, $locstr, $lit);
        IF nfl.flastr THEN freestring( nfl.flastr := 0,
        $dspblk);
        nfl.flastr _ getstring( nfilnam.L + 1, $dspblk);
        *[nfl.flastr]* _ *nfilnam*;
        !JSYS gjinf;
        %-NSW%                                              5X2D1V13
           temp _ R1;  %loggin dir%
        %-NSW%                                              5X2D1V14
        %+NSW%                                              5X2D1V15
           temp _ R2;  %connected dir%
        %+NSW%                                              5X2D1V16
        IF nfl.flpcst THEN freestring( nfl.flpcst := 0,
        $dspblk);
        nfl.flpcst _ cpcnam (nfl.flastr, temp);
     %propagate privacy status%
        chprvsts (nfn, oprvsts);
     % propagate archive "don't archive" and "don't delete%
        !gtfdb(ofl.florig, 1000017B, $R3);
        !chfdb(17B6 .V njfn, 11B10); %R3 set by gtfdb%
     IF NOT writeout(ofn,$lit) OR NOT delpc (ofn, $lit) THEN
     ABORT (badupdate, $lit);
     IF lit.L > empty THEN
        BEGIN
        dismes (2, $lit);
        END;
        %delete partial copy from user's directory and unlock
        original file (if it exists)%
     % close old file %
        dumpc(nfn, FALSE);  %turn off pc bits in header%
        IF NOT clsfil (ofn, $lit) THEN ABORT (badupdate,
        $lit);
     %make new file the original%
        nfl.florig _ nfl.flpart := 0;
        filepart[nfn] _ FALSE;
     % make new file use old file number %
        % write out all pages from the file %
        IF NOT writeout (nfn, $lit) THEN ABORT (badupdate,
        $lit);
        cpyfle (nfn, ofn);
           %copies new file list entry to old file list
           entry%
        clrfil (nfn); %clear  file list entry for new file%
        IF NOT rdhdr(ofn, $lit) THEN err ($lit);
        clpsid($clhead); %convert new sid's back to psid's in
        correspondence list%
        clupdt();  %done here so file header can be mapped
        out after markers are updated per new psid's%
        R1.LH _ 485;  R1.RH _ filhdr(ofn)/512;
        R2.LH _ njfn;  R2.RH _ 0;
```

```
            p2 _ 0;
            !JGVS pnap;
            p2 _ 1I;
            p1 _ -1;
            !JGVS pnap;
            % close and re-open outputed file with read-only
            status %
               IF NOT clsfil (ofn, $lit) THEN ABORT (badupdate,
               $lit);
               IF NOT opnfil (ofn, $lit) THEN ABORT (badupdate,
               $lit);
               IF lit.L > empty THEN
                  BEGIN
                  dismes (2, $lit);
                  END;
         %delete old versions%
            delovsrns (f[intadr(ofn)].[lorig, nvtk);
         skpachk _ curskpachk;
         DROP(sigassign);
         RETURN;

         (sigassign) CATCHPHRASE;                               5X2D1AH
            BEGIN
            DISABLE (sigassign);
            CASE SIGNALTYPE OF
               = aborttype: skpachk _ curskpachk;
            ENDCASE;
            CONTINUE;
            END;

         END.

      (setrot)    % LS: used to build statements %
      PROCEDURE (ptr, fnstng, dtstng % => no value %);          5X2D2
         % Procedure description
            FUNCTION
               none
            ARGUMENTS
               none
            RESULTS
               none
            NON-STANDARD CONTROL
               none
            GLOBALS
               none
            EXAMPLE
               none
            %
         % Declarations %
            REF ptr, fnstng, dtstng;
         FIND SP(ptr) ^p1 ^p2 ["; ; ; ;"] ^p2;
         IF FIND < [" >>>"] > 4CH ^p1
         THEN ST p1 p2 _ *fnstng*, ", " , *dtstng*, SP, *initsr*,
         " ; ; ; ;"
         ELSE
            IF NOT getnmf(ptr)
```

```
            THEN ST ptr _ *fnstng*, ", " , *dtstng*, SP,
            *initsr*, " ;;;;", p2 SE(ptr);
         RETURN;
         END.


      %.....update and compact file support routines.....%
         (copfil) PROCEDURE %***% % Copies and compacts ring and
         data blocks of files (used by outfile) %                5X2E1
            (oldfil, % NLS file numebr of the old file being output
            %
            newfil,  % NLS file numebr of the new file being output
            to %
            list     % Address of correspondence list head %
            );
            %This routine copies the contents of the file identified
            by oldfil to the file identified by newfil, one
            statement at a time, using clsid to convert the entries
            in the correspondence list to sid's and clpsid to
            convert the sid's to the new stid's.  The correspondence
            list is asssumed to contain stid's of interest, such as
            the csp's and frozen list stid's for each da, entries on
            the link-jump rings, and markers for this file.%
            %-----------------%
            LOCAL
               lstid,     %last statement identifier%
               ostid,     %old statement identifier%
               nstid,     %new statement identifier%
               nhd,       %new file header address%
               ohd,       %old file header address%
               mkrlen,    %number of entries in marker table%
               oosdb,     %old origin sdb%
               dir;       %direction%
            LOCAL TEXT POINTER z1, z2, z3, z4;
            LOCAL STRING nfnam[50], ofnam[50];
            REF list, oosdb;
            ostid _ nstid _ lstid _ 0;
            ostid.stfile _ oldfil;
            lstid.stfile _ nstid.stfile _ newfil;
            ostid.stpsid _ lstid.stpsid _ nstid.stpsid _ origin;
            intfil(newfil);
               %initialize new file header%
            LOOP  %copy statements from old file to new%
               BEGIN
               IF (nstid _ getsub(ostid)) NOT= ostid THEN
                  BEGIN  %statement has sub structure%
                  ostid _ nstid;
                  dir _ sub;
                  END
               ELSE  %no substructure--check if tail%
                  BEGIN
                  IF ostid.stpsid = origin THEN EXIT;
                  IF NOT getftl(ostid) THEN
                     BEGIN  %not tail--get successor%
                     ostid _ getsuc(ostid);
                     dir _ suc;
```

```
            END
         ELSE  %tail--get successor of highest "up"
         proccessed%
            BEGIN
            DO
               BEGIN  %get "up" until not tail%
               IF (ostid _ getup(ostid)).stpsid = origin
               THEN
                  EXIT 2;
               lstid _ getup(lstid);
               END
            UNTIL NOT getftl(ostid);
            dir _ suc;
            ostid _ getsuc(ostid);
            END;
         END;
      nstid _ newrng(newfil);
         %allocate new ring block in new file%
      coprng(ostid, nstid);
         %copy pertinent feilds from old ring block to new%
      %insert new ring element into new file
         as suc or sub of lstid%
      IF dir = suc THEN <STRMNP, inss> (lstid, nstid,
      nstid)
      ELSE <STRMNP, insd> (lstid, nstid, nstid);
      copplist(ostid, nstid);
         %copy contents of oldsdb to new%
      lstid _ nstid;
      END;
   *ofnam* _ NULL;
   *nfnam* _ NULL;
   filnam(oldfil, $ofnam);
   filnam(newfil, $nfnam);
   uplsfnm($ofnam, $nfnam); %update file name in link
   stack%
   %replace origin statement%
      p1 _ p2 _ 0;
      p1.stfile _ oldfil;
      p2.stfile _ newfil;
      p1.stpsid _ p2.stpsid _ origin;
      p1[1] _ p2[1] _ 1;
      FIND SF(p2) ^z1 SE(p2) ^z2 SF(p1) ^z3 SE(p1) ^z4;
      creptex($z1, $z2, $z3, $z4);
      stosid(p2, getsid(p1));
      lodprop(p1, txttyp : &oosdb); %get old origin sdb
      and%
      csetnsta(p2, oosdb.slnmdl, oosdb.srnmdl); %copy over
      name delims%
   %copy sidcnt from old header%
      [(nhd _ filhdr(newfil)) + $sidcnt-$filhed] _
         [(ohd _ filhdr(oldfil)) + $sidcnt-$filhed];
   %copy default link directory from old header%
      [nhd  + $funo-$filhed] _
         [ohd + $funo-$filhed];
   %copy marker table%
      mkrlen _ [ohd + $mkrtbl-$filhed];
```

```
            [nhd + $mkrtbl-$filhed] _ mkrlen;
            IF mkrlen > 0 THEN
               mvbfbf(ohd + $mkrtb-$filhed, nhd + $mkrtb-$filhed,
               mkrlen*mkrl);
         %set lwtim, finit%
            [nhd + $finit - $filhed] _ cinit;
            [nhd + $lwtim - $filhed] _ gtadcall();
         RETURN;
         END.

      (coprng) %***% PROCEDURE %Copy name flag, name hash, and
      SID fields from one ring element to another %  % Since this
      is called right before a call on copsdb in copfil (the only
      call of this) and since copsdb does these copies except for
      SID (with good reason since it is used in other contexts),
      we could elimiate much of this code. %                    5X2E2
         (ostid, % stid of source ring %
         nstid   % stid of destination ring %
         );
         %This routine copies the ring element, ostid, to nstid.%
         %-----------------%
         LOCAL pgnum, oring, nring;
         REF oring, nring;
         pgnum _ lodent(ostid, rngtyp : &oring);
         frzblk(pgnum, 1);
         INVOKE (sigrlse);
         lodent(nstid, rngtyp : &nring);
         nring.rnamef _ oring.rnamef;
         nring.rtorgin _ oring.rtorgin;
         nring.rnameh _ oring.rnameh;
         nring.rsid _ oring.rsid;
         frzblk(pgnum, -1);
         DROP (sigrlse);
         RETURN;

         (sigrlse) CATCHPHRASE;                                 5X2E2T
            BEGIN
            DISABLE (sigrlse);
            frzblk(pgnum, -1);
            CONTINUE;
            END;

         END.

%verify%
   %verify PCP interface routine%
      (xverify) %Execute Verify Command%                          5Y1A
         PROCEDURE
            %FORMALS%
               (window);     %window number%                    5Y1A1A1
         %--------------------%
         vfmain(dspfile(window), TRUE);
         dismes (2, $"Successful: internal structure is OK");
         RETURN;
         END.
%viewspecs from the mouse buttons%
```

```
%mouse button viewspecs PCP interface routine%
   (xmousespecs) %Execute Mouse Viewspecs%
   PROCEDURE (                                                    5Z1A
      %FORMALS%
         vs REF,          %viewspec chars typed in%
         window);         %number of last window bugged%
      LOCAL da REF, vsstr REF, char, save, frmt, extravs, i,
      curvs[2];
      LOCAL STRING locstr[20];
      REF vswndw, ttyda;
      %--------------------%
      IF inhelp THEN RETURN;  %no mouse viewspecs in Help%
      dspccf _ FALSE;
         %don't clear command feedback window -- could be in
         middle of command%
      &vsstr _ &vs + 4;  %viewspec string starts in 5th word of
      viewspec block%
      &da _ dsparea(window);
      (curvs, curvs[1]) _ (da.davspec, da.davspc2);
      FOR i _ 1 UP UNTIL > vsstr.L DO
         CASE (char _ *vsstr*[i]) OF
            = 'F :  %recreate the display using dafrmt%
               BEGIN
               % clear status (tty) window %
                  dfeclw(ttyda.dawid);
               extravs _ FALSE;
               IF curvs.vsrlev THEN
                  curvs _ reslev(curvs, getlev(da.dacsp));
               da.dapvs _ da.davspec _ cspvs _ curvs;
               da.dapvs2 _ da.davspc2 _ cspvs[1] _ curvs[1];
               save _ da.davspec.vsdaft := TRUE;
               dafrmt (&da, 0);
               da.davspec.vsdaft _ save;
               IF NOT save THEN extravs _ TRUE;
               END;
            = 'f :  %recreate the display using daupdate%
               BEGIN
               % clear status (tty) window %
                  dfeclw(ttyda.dawid);
               extravs _ FALSE;
               IF curvs.vsrlev THEN
                  curvs _ reslev(curvs, getlev(da.dacsp));
               da.davspec _ cspvs _ curvs;
               da.davspc2 _ cspvs[1] _ curvs[1];
               save _ da.davspec.vsdaft := TRUE;
               dpset(dspjpf, endfil, endfil, endfil);
               IF howformat(&da, da.dacsp.stfile, endfil : frmt)
               THEN
                  IF frmt THEN dafrmt(&da, 0)
                  ELSE daupdate (&da);
               da.davspec.vsdaft _ save;
               IF NOT save THEN extravs _ TRUE;
               da.dapvs _ da.davspec;
               da.dapvs2 _ da.davspc2;
               END;
            ENDCASE
```

```
                BEGIN
                extravs _ TRUE;
                curvs _ setlt(char, curvs, curvs[1]: curvs[1]);
                END;
          IF extravs THEN
             BEGIN
             bldvsstr(curvs, curvs[1], $locstr);
             wrlstr(&vswndw, $locstr, FALSE);
             da.davspec _ cspvs _ curvs;
             da.davspc2 _ cspvs[1] _ curvs[1];
             holdvs _ 1;  %global to keep cmdfinish from doing too
             much too soon%
             END;
          RETURN;
          END.

FINISH of psedt2                                                    6
   (substatus) PROCEDURE (                                          6A

      % The status of a substitute command is printed out before
      execution, given the entity type being substituted%
      % FORMAL ARGUMENTS %
         type REF);              % type of selection made %
      LOCAL pairstr REF, cardp, wbp1, wbp2, i, j, typstr REF, colwidth;
      LOCAL TEXT POINTER tp1, tp2;
      LOCAL STRING between[10], statstr[500], sentstr[10],
      tempstr[120], oldstr[120], newstr[120];
      LOCAL LIST pcplist[2];
      POINTER cardp;


      %Set up the table heading for output%
      colwidth _ 20;  %width of a column for non-text substitutions%
      &typstr _ ELEM #type#[tppair];
      tp1 _ typstr;
      tp2 _ typstr[d2sel];
      *sentstr* _ tp1 tp2;
      FOR j _ sentstr.L UP UNTIL = sentstr.M DO
         *sentstr* _ *sentstr*,SP;
      *statstr* _ "   SUBSTITUTE ", *sentstr*, CR, LF;
      IF *sentstr* = "TEXT      " THEN
         *between* _ CR, LF, "OLD:  "
      ELSE
         BEGIN
         *statstr* _ *statstr*, "NEW ", *sentstr*, "        OLD ",
         *sentstr*, CR, LF, CR, LF;
         *between* _ "  ";
         END;

      %Get ahold of the packed string of pairs%
      &pairstr _ subhed.sbas;

      i _ 1;              %Index through the string%
      WHILE i < pairstr.L DO
         BEGIN
         oldstr.L _ newstr.L _ cardp _ 0;
```

```
    cardp _ subhed.sbdp + *pairstr*[i];
       %get pointer to chain describing entities staring with the
       first character of this entity%
    IF [cardp] = 0 THEN ABORT(badsub,$"substatus:  Substitution
    array bad.")
    ELSE cardp _ [cardp];
    DO
       BEGIN
       wbp1 _ chbmty + $oldstr;
       wbp2 _ cardp.catbp;
       FOR j _ 0 UP UNTIL = cardp.catnc DO
          BEGIN
          IF j = oldstr.M THEN EXIT;
          ^wbp1 _ ^wbp2;   %store the old string%
          END;
       oldstr.L _ MIN(cardp.catnc, j);
       *tempstr* _ *pairstr*[i TO oldstr.L+i-1];
       (comstrs):                                                  6A23F7
       IF *tempstr* = *oldstr* THEN
          BEGIN    %This card describes the next pair%
          wbp1 _ chbmty + $newstr;
          wbp2 _ cardp.carbp;
          FOR j _ 0 UP UNTIL = cardp.carnc DO
             BEGIN
             IF j = newstr.M THEN EXIT;
             ^wbp1 _ ^wbp2;        %store the new string%
             END;
          newstr.L _ MIN(cardp.carnc, j);
          EXIT;
          END
       ELSE cardp _ cardp.canxt;   %next entry%
       END
    WHILE cardp;

    IF oldstr.L = 0 THEN ABORT(badsub,$"substatus:  String not
    found in array.");

    %Add this entry to output status string%
    FOR j _ newstr.L UP UNTIL >= colwidth DO
       *newstr* _ *newstr*, SP;
    IF *sentstr* = "TEXT      " THEN *statstr* _ *statstr*, CR,
    LF, "NEW:  ";  %blank lines between pairs of text%
    *statstr* _ *statstr*, *newstr*, *between*, *oldstr*, CR, LF;

    %Increment i for next pair of strings%
    i _ i + cardp.catnc + cardp.carnc;
    END;

%Send it out and set display globals%
#pcplist# _ USE makedesc(ustring, $statstr, FALSE), USE
makedesc(uboolean, cawait, FALSE);
pcpcall (fenrh, fepkh, $"show", $pcplist, 0);
dpset(dspno, endfil, endfil, endfil);

RETURN;
END.
```

```
(cprint) %***% PROCEDURE (da, stid1, stid2, type, coretflag, oldsw);
                                                                    68
    %print the structure (type) addrssed by stid1,stid2;  oldsw is
    the address of an already open sequence work area or zero.  If
    zero, a sequence work area is opened by printg and closed;  if
    non-zero, the sequence is not closed by printg. %
    %-----------------%
    LOCAL
       i, %control variable for loop%
       contsw,  %continue returning more statements%
       char,    %current character%
       gapcol, %column of char preceding last invisible char%
       gapptr, %pointer to char preceding last invisible char%
       startp, %pointer to start of line character%
       prindt,  %number of columns to indent%
       lincnt, %count of lines printed for current statement%
       stid,    %current stid being printed%
       maxcol, %maximum number of columns for current line%
       stnrt,  %whether stmt numbers are to be printed on right%
       extraline,  %true if stmt number put on separate line%
       stnlength,  %save length of stmt no.%
       sw,      %address of sequence generator work area%
       top,     %counter for spacing at top of page%
       vspc1,  %viewspecs from sequence work area %
       vspc2,
       head,    %address of file header%
       cntrlstr, %address of non-printing char string%
       sig2, sig3, sig4,      %signal results%
       mkrptr; %marker table pointer%
    LOCAL STRING
       str[100],     %scratch string%
       stnsig[50];  %holds statement number or signature%
    LOCAL TEXT POINTER z1;
    LOCAL LIST tempargs[4], colist[1], crtnlist[1];
    REF oldsw, sw, da, mkrptr;

    IF stid1 = endfil THEN ABORT(eof,$"End of file.");
    da.dacrow _ 0;  %start a page for later pagination%
    IF da.davspec.vspagf THEN
       BEGIN
       top _ 3;
       UNTIL (top := top - da.davinc) <= 0 DO da.dacrow _ da.dacrow +
       da.davinc;
       END;
    pageno _ 0;
    &sw _ IF &oldsw THEN &oldsw
       ELSE openseq (stid1, IF da.davspec.vsbrof THEN stid1 ELSE
       stid2 % olex only should be handled by the calling routine
       since stid1 and stid2 supposedly form a legitimate group! %,
       da.davspec, da.davspc2, da.dausgcod, da.dacacode);
    INVOKE (closesw);
    z1[0] _ IF stid1.stastr THEN da.dacsp ELSE stid1;

    UNTIL ((stid _ seqgen (&sw)) = endfil) OR
      (type = stmtv AND sw.swcstid # stid1) OR (NOT contsw) DO
       BEGIN
```

```
% get viewspecs from sequence work area. %
   vspc1 _ da.davspec _ sw.swvspec;
   vspc2 _ da.davspc2 _ sw.swvsp2;
%set up work area s2work for READC%
   s2work _ stid;
   s2workflg _
      IF vspc1.vsnamf OR stid.stastr THEN 1 %print name%
      ELSE fchtxt (stid); %skip past name%
   fechc1 (forward, $s2work);
%markers (not allowed for now)%
   prmkrf _ 0;
   IF FALSE AND vspc1.vsmkrf AND NOT stid.stastr THEN
      BEGIN  %see if marker in statemnt%
      &mkrptr _ $mkrtb - $filhed +
         (head _ filhdr(stid.stfile));
      mkrend _ &mkrptr +
         [$mkrtbl - $filhed + head] * mkrl;
      mkrflg _ FALSE;
      FOR mkrptr UP mkrl UNTIL = mkrend DO
         IF mkrptr.mkpsid = stid.stpsid THEN
            BEGIN
            mkrflg _ TRUE;
            mkrcnt _ mkrptr.mkccnt + 1;
            EXIT;
            END;
      END;
prindt _          %indentation%
   CASE TRUE OF
      = vspc1.vsindf:
         MAX (tpoffset, MIN (da.daind * (sw.swclvl-1) +
         tpoffset, da.damind, spacestr.M));
      = vspc1.vsbrof, = vspc1.vsplxf:
         MAX (tpoffset, MIN (da.daind * (sw.swclvl-sw.swslvl)
         + tpoffset, da.damind, spacestr.M));
      ENDCASE tpoffset;
   % TPOFFSET is a global which the user can set via Execute
   Viewchange.  This allows the user to control the left
   margin of his print out. %
*lit* _ *spacestr* [empty + 1 TO prindt];
maxcol _ da.damcol -2;   %max no. cols%
stnrt  _ FALSE;
IF vspc1.vsstnf AND stid.stpsid # origin AND
 NOT stid.stastr THEN
   IF NOT vspc1.vsstnr THEN
      BEGIN  %print statement number%
      IF vspc1.vssidf
         THEN % display sid's %
            *lit* _ *lit*, '0, STRING( getsid(stid) )
         ELSE % display line numbers %
            fechnm (sw.swsvw, $lit);
      *lit* _ *lit*, SP;
      END
   ELSE   % statement numbers go on the right %
      BEGIN
      *stnsig* _ NULL;
      IF vspc1.vssidf
```

```
            THEN % display sid's %
               *stnsig* _ '0, STRING( getsid(stid) )
            ELSE % display line numbers %
               fechnm (sw.swsvw, $stnsig);
         stnlength _ stnsig.L;
         stnrt  _ TRUE;
         END;
      gapcol _ da.daccol _ lit.L * da.dahinc;
   %line printing loop%
   FOR lincnt _ 0 UP UNTIL = vspc1.vstrnc DO
      BEGIN
      IF inptrf THEN EXIT 2;
      gapptr _ startp _ lit.L;
      UNTIL da.daccol >= maxcol DO
         CASE char _ READC ($s2work) OF
            =SP:
               BEGIN
               gapptr      _ lit.L;
               gapcol      _ da.daccol;
               da.daccol _
               da.daccol+putchr(&da,char,da.daccol,$lit);
               END;
            =ENDCHR, =EOL, =CR:
               BEGIN
               gapptr      _ lit.L;
               gapcol      _ da.daccol;
               EXIT LOOP;
               END;
            =TAB:
               BEGIN
               gapptr      _ lit.L;
               gapcol      _ da.daccol;
               da.daccol _
               da.daccol+putchr(&da,char,da.daccol,$lit);
               END;
            =CA, =C., =LF, =CD, =BC, =BW, =$ascalt, IN [1B,32B],
            IN [34B,36B]: %an acceptable non-printing character%
               BEGIN
               cntrlstr _ npstrad(char);%get np representation
               (address)%
               %see if non-printing character string will exceed
               current line%
               IF [cntrlstr].L + da.daccol >= maxcol THEN
                  BEGIN
                  s2work[1] _ s2work[1] - 1;
                  fechc1 (forward, $s2work);  %repeat last char
                  next time through%
                  REPEAT CASE (char _ EOL);
                  END
               ELSE %fits on current line%
                  FOR i _ 1 UP UNTIL > [cntrlstr].L DO
                     BEGIN
                     char _ *[cntrlstr]*[i];
                     da.daccol _
                     da.daccol+putchr(&da,char,da.daccol,$lit);
                     END;
```

```
                END;
            ENDCASE
                BEGIN
                da.daccol _
                da.daccol+putchr(&da,char,da.daccol,$lit);
                END;
        %by here, a line has been constructed. fix gapptr and
        gapcol if there were no invisibles%
            IF startp = gapptr THEN   %no invisibles in line%
                BEGIN
                gapptr _ lit.L;
                gapcol _ da.daccol;
                END;
        IF (char = ENDCHR OR lincnt = vspc1.vstrnc) AND stnrt AND
        da.daccol + stnsig.L + 2 <= maxcol THEN extraline _ FALSE
            % can cram statement number on right of this line (which
            is the last line of the statement), so don't print it
            out yet%
        ELSE
            BEGIN
            extraline _ TRUE;
            IF NOT (char = ENDCHR OR lincnt +1 = vspc1.vstrnc) THEN
                *lit* [prindt + 1 TO lit.L] _ *lit* [gapptr + 2 TO
                lit.L]
            ELSE
                *lit* _ *spacestr*[empty + 1 TO prindt];
            gapcol _ da.daccol _ lit.L*da.dahinc;
            END;
        z1[0] _ stid;
        IF char = ENDCHR THEN EXIT;
        END;

    % The statement is completed.%

    %statement numbers on right%
        IF stnrt THEN
            BEGIN
            IF extraline THEN
                BEGIN          %putting number on separate line%
                *lit* _ *lit*, EOL, *spacestr*[1 TO maxcol -
                stnsig.L-2];
                END
            ELSE
                BEGIN
                *lit* _ *lit*, *spacestr*[1 TO maxcol - da.daccol -
                stnsig.L -2];
                END;
            *lit* _ *lit*, SP,SP, *stnsig*;
            END
        ELSE extraline _ FALSE;
    %blank line and signature%
        IF vspc1.vsidtf AND NOT stid.stastr THEN
            BEGIN
            *stnsig* _ NULL;
            fechsig (stid, $stnsig);
            IF extraline AND stnlength + stnsig.L +3 > maxcol -
```

```
            prindt THEN
               extraline _ FALSE;
            IF extraline THEN *lit*[maxcol - stnlength - stnsig.L -2
            TO maxcol - stnlength -3] _ *stnsig*
            ELSE *lit* _ *lit*, EOL,
               *spacestr* [1 TO maxcol - stnsig.L], *stnsig*;
            END;
      IF coretflag = cortn THEN
         BEGIN
         %Temporarily return the statement directly to the Frontend%
         #colist#[1] _ USE rtnstring($lit);
         contsw _ coreturn ($colist, $crtnlist);
         contsw _ ELEM #crtnlist#[1];
         END;
      END;
   IF NOT &oldsw THEN closeseq (&sw);
   DROP (closesw);
   RETURN;

   %Define the catchphrase, closesw%

   (closesw) CATCHPHRASE (:sig2, sig3, sig4);                          6B24
      BEGIN
      DISABLE (closesw);
       IF NOT &oldsw THEN closeseq(&sw);
      END;

   END.
%...moved to here by ROM ...%
   (csimdev)    %Execute simulate Device Command%                      6C1
      PROCEDURE(da, devicetype, modetype);
      REF da;
      REF rawchr, msgda, tda;
      LOCAL oldnlmode; %for saving ced entry nlmode setting
                        (changed by setdev)%
      oldnlmode _ nlmode;
      IF devicetype NOT= nldevice OR modetype NOT= nlmode THEN
         BEGIN
         continue _ TRUE;
         IF modetype NOT= nlmode THEN
            BEGIN
            IF nlmode = fulldisplay THEN shutdis();
            setdev (devicetype);
            initch(devicetype);
            initbtbl();
            IF oldnlmode = typewriter THEN % TNLS --> DNLS %
               BEGIN
               IF savedda THEN  %came from DNLS before -- restore
               old da%
                  BEGIN
                  &tda _ savedda := 0;
                  tda.dasuppress _ FALSE;
                  tda.dacsp _ da.dacsp;
                  tda.daccnt _ 1;
                  tda.davspec _ da.davspec;
                  tda.davspc2 _ da.davspc2;
```

```
                tda.dalink _ da.dalink := tda.dalink;
                tda.dafrzl _ da.dafrzl := 0;
                tda.dapstf _ da.dapstf;
                tda.dacacode _ da.dacacode;
                tda.dausqcod _ da.dausqcod;
                tda.daukeycod _ da.daukeycod;
                delda(&da := 0);
                END
            ELSE  %first time%
                BEGIN
                continue _ FALSE;
                savetda _ &da;
                &tda _ newda();
                intdafl (&tda);
                tda.dacsp _ da.dacsp;
                tda.daccnt _ 1;
                tda.davspec _ da.davspec;
                tda.davspc2 _ da.davspc2;
                tda.dalink _ da.dalink := tda.dalink;
                tda.dafrzl _ da.dafrzl := 0;
                tda.dapstf _ da.dapstf;
                tda.dacacode _ da.dacacode;
                tda.dausqcod _ da.dausqcod;
                tda.daukeycod _ da.daukeycod;
                da.dasuppress _ TRUE;
                inittimer();
                END;
            END
        ELSE % DNLS --> TNLS %
            BEGIN
            IF savetda THEN  %came from TNLS before -- restore
            old da%
                BEGIN
                &tda _ savetda := 0;
                tda.dasuppress _ FALSE;
                tda.dacsp _ da.dacsp;
                tda.daccnt _ 1;
                tda.davspec _ da.davspec;
                tda.davspc2 _ da.davspc2;
                tda.dalink _ da.dalink := tda.dalink;
                tda.dafrzl _ da.dafrzl := 0;
                tda.dapstf _ da.dapstf;
                tda.dacacode _ da.dacacode;
                tda.dausqcod _ da.dausqcod;
                tda.daukeycod _ da.daukeycod;
                delda(&da := 0);
                END
            ELSE  %first time%
                BEGIN
                savedda _ &da;
                &tda _ newda();
                intdafl (&tda);
                tda.dacsp _ da.dacsp;
                tda.daccnt _ 1;
                tda.davspec _ da.davspec;
                tda.davspc2 _ da.davspc2;
```

```
                tda.dalink _ da.dalink := tda.dalink;
                tda.dafrzl _ da.dafrzl := 0;
                tda.dapstf _ da.dapstf;
                tda.dacacode _ da.dacacode;
                tda.dausqcod _ da.dausqcod;
                tda.daukeycod _ da.daukeycod;
                da.dasuppress _ TRUE;
                END;
             vspsav _ vspsav[1] _ 0; %viewspecs will be refreshed
             upon reentry%
             END;
          END;
       initdis();
       continue _ FALSE;
       IF nlmode = fulldisplay THEN dsubsys( $ssysname );
       END;
    RETURN;
    END.

(xsimulate) %Execute Simulate Command%                          6C2
    PROCEDURE
       %FORMALS%
          (devicetype);    %device type%                       6C2A1A
       LOCAL dev, modetype;
    %----------------------%
    dev _ strdev(devicetype :modetype);
    csimdev(!da(), dev, modetype);
    cspupdate _ FALSE;
    IF NOT nodisplay THEN dpset(dspallf, endfil, endfil, endfil);
    RETURN;
    END.
```

(checkmore)	<nine, pshelp, 01907>	PROCEDURE	4A
(chkdsp)	<nine, pshelp, 0692>	PROCEDURE	7A
(clohtil)	<nine, pshelp, 0634>	PROCEDURE	7B
(conbck)	<nine, pshelp, 0810>	PROCEDURE	8A
(conint)	<nine, pshelp, 03034>	PROCEDURE	7C
(conink)	<nine, pshelp, 01957>	FIELD - 36	2F1
(constd)	<nine, pshelp, 01951>	FIELD - 36	2E1
(constk)	<nine, pshelp, 01952>	FIELD - 18	2E2
(contrg)	<nine, pshelp, 01950>	RECORD	2E
(didntwork)	<nine, pshelp, 01887>	CATCHPHRASE	3D
(feedsw)	<nine, pshelp, 01808>	PROCEDURE	9A
(freemenu)	<nine, pshelp, 03020>	PROCEDURE	7D
(help2show)	<nine, pshelp, 02891>	PROCEDURE	5E
(helpinit)	<nine, pshelp, 02594>	PROCEDURE	5A
(helpring)	<nine, pshelp, 02769>	PROCEDURE	5C
(helpshow)	<nine, pshelp, 02856>	PROCEDURE	5D
(helpterm)	<nine, pshelp, 03184>	PROCEDURE	5B
(mencont)	<nine, pshelp, 01694>	PROCEDURE	9B
(menustart)	<nine, pshelp, 0772>	PROCEDURE	8B
(moreterm)	<nine, pshelp, 0624>	PROCEDURE	7E
(numist)	<nine, pshelp, 0783>	PROCEDURE	8C
(ofstid)	<nine, pshelp, 01796>	PROCEDURE	9C
(pushent)	<nine, pshelp, 01833>	PROCEDURE	10A
(qappender)	<nine, pshelp, 01321>	PROCEDURE	9D
(qcolumnate)	<nine, pshelp, 01736>	PROCEDURE	9E
(qdirparse)	<nine, pshelp, 01347>	PROCEDURE	9F
(qdisp)	<nine, pshelp, 0524>	PROCEDURE	7F
(qgetup)	<nine, pshelp, 0717>	PROCEDURE	8D
(qinclude)	<nine, pshelp, 01382>	PROCEDURE	9G
(qinkspec)	<nine, pshelp, 01763>	PROCEDURE	9H
(qmenu)	<nine, pshelp, 01548>	PROCEDURE	9I
(qsearch)	<nine, pshelp, 0730>	PROCEDURE	8E
(qsender)	<nine, pshelp, 01306>	PROCEDURE	9K
(qsparse)	<nine, pshelp, 01454>	PROCEDURE	9L
(qspcreset)	<nine, pshelp, 01499>	PROCEDURE	9M
(qstmt)	<nine, pshelp, 01261>	PROCEDURE	9N
(qstrinit)	<nine, pshelp, 0581>	PROCEDURE	7G
(qstrip)	<nine, pshelp, 01511>	PROCEDURE	9O
(quercol)	<nine, pshelp, 01961>	FIELD - 1	2G2
(querinc)	<nine, pshelp, 01960>	FIELD - 1	2G1
(queropts)	<nine, pshelp, 01963>	FIELD - 8	2G4
(querprt)	<nine, pshelp, 01962>	FIELD - 1	2G3
(queryseq)	<nine, pshelp, 01215>	PROCEDURE	9J
(quspecs)	<nine, pshelp, 01959>	RECORD	2G
(qvalid)	<nine, pshelp, 0819>	PROCEDURE	8F
(rstmore)	<nine, pshelp, 0378>	PROCEDURE	4B
(samename)	<nine, pshelp, 0979>	PROCEDURE	8G
(samevw)	<nine, pshelp, 0992>	PROCEDURE	8H
(srchfndx)	<nine, pshelp, 01014>	PROCEDURE	8I
(stkcnt)	<nine, pshelp, 01958>	FIELD - 5	2F2
(stkhd)	<nine, pshelp, 01956>	RECORD	2F
(stktyp)	<nine, pshelp, 01955>	FIELD - 3	2E4
(totcnt)	<nine, pshelp, 01954>	FIELD - 15	2E3
(tstmenu)	<nine, pshelp, 01711>	PROCEDURE	9P
(wbranm)	<nine, pshelp, 01178>	PROCEDURE	8J
(wbrsrh)	<nine, pshelp, 01070>	PROCEDURE	8K

(wfndit)	<nine, pshelp, 0831>	PROCEDURE	8L
(wgetadr)	<nine, pshelp, 0892>	PROCEDURE	8M
(wgetfile)	<nine, pshelp, 0878>	PROCEDURE	8N
(wprstile)	<nine, pshelp, 0854>	PROCEDURE	8O
(wstrtnm)	<nine, pshelp, 0920>	PROCEDURE	8P

```
< NINE, PSHELP.NLS;22, >, 12-Aug-78 18:52 BLP ;;;;
FILE pshelp % (arcsubsys, L109,) to (relnine, pshelp.rel,) %
% HELP declarations not copied over in NLS 9 %
   DECLARE                  % From <nls, fconst,> %
      stkmax = 26,
      entind = -1,
      rngmax = 26,
      rnglnt = 2;
   % DECLARE                 From <nls, fdata,> %
      % athelp_0,            TRUE if at help command, FALSE otherwise. %
      % inhelp_0;            TRUE if in help command, FALSE otherwise. %
   DECLARE                  % From <nls, fdata,> %
      fwrd,                 % true if the first word of a statement bugged
      %
      glblstd;              % the stid of the last branch search %
   DECLARE STRING           % From <nls, fdata,> %
      % failnm[100],        name searched for but failed %
      defvspc[100];         % default viewspecs for a file %
   (contrg) RECORD          % a context ring item-- two words long %      2E
      constd[36],           % stid of principal node of this context %
      constk[18],           % pointer to top of menu stack for this
      context.
         It will be zero if unused %
      totcnt[15],           % total number of items in the stack %
      stktyp[3];            % type of stack-- menu, multiple, unused %
   (stkhd) RECORD           % 0th element of context stack block %        2F
      conlnk[36],           % link to next block in this stack;
         -1 if unlinked, 0 if a free block %
      stkcnt[5];            % number of elements in this block (<= 25) %
   (quspecs) RECORD         % Query directives embedded in data base %    2G
      querinc[1],           % Include substructure at node in the menu. %
      quercol[1],           % During printout, "columnate" all menu items.
      %
      querprt[1],           % Print QUERY viewspecs, links for debugging %
      queropts[8];          % Sets number of options printed at one time. %

% DECLARATIONS %
   DECLARE EXTERNAL helpfirstfile, menusw, qwa;
   REF menusw;     % global menu sequence generator -- only ONE %
   REF inpt, conrng, qda, qwa, cda, qsw, curstk, qstorblk, qnewstmt,
   hipcmdstk;
   (didntwork) CATCHPHRASE ();                                          3D
      CASE SIGNALTYPE OF
         = aborttype: TERMINATE;
         ENDCASE CONTINUE;
   NUMESS;

% HELP Parsefunctions %
   (checkmore) PROCEDURE (values REF LIST);                             4A
      % dav: checks the value of the global variable 'moremen' %
         #values#[1] _ moremen;
         RETURN;
         END.

   (rstmore) PROCEDURE;                                                 4B
      % Reset moremen (did not want next menu); close sequences.
```

```
    Moremen always FALSE, sequence qsw closed upon exit from this
    rule. %
        moremen _ FALSE;
        IF &qsw THEN moreterm();
            % Must be at a lower level because of symbol conflicts.
            Closes sequences and resets viewspecs. %
        RETURN;
        END.

% Help execution functions. %
  (helpinit)   % Initializes the HELP system %
  PROCEDURE (tool REF, context REF % not used %, helplink REF);      5A
    % Procedure description
        FUNCTION
            Does the initialization required upon entry into the help
            command.  It puts out a help message, loads the help file
            if necessary, does initialization and allocation of
            storage.  Initializes moremen to FALSE (no more to be
            shown).
        ARGUMENTS
            tool - STRING - the name of the tool we are currently in
                Example: "BASE"
            context - STRING -  not used
            helplink - LIST - an LSEL literal pointing to the user's
            typein or eventually the command context when the HELP
            button was hit.  There will be a builtin HELPSTRING which
            will have the appropriate context.  Or NULL if at the
            herald
                Example: ([INTEGER 30]  [LITERAL foo]  [INTEGER 1])
        RESULTS
            None
        NON-STANDARD CONTROL
            SIGNALS GENERATED
                err($"No help has been provided for <toolname>")
                    if the tool does not have a HELP file, and if the
                    user does not explicitly type one, and if we can't
                    find the default file NLS.NLS
                err($"I am the elusive Help command bug:
                tell FEEDBACK details of what you were doing.")
                    if any of a variety of low-level system errors occur
            SIGNALS CAUGHT
                All ABORT signals are terminated.
        GLOBALS
            Variables
                calcaux, dacnt, dal, defvs1, defvs2, dpyarea, dspno,
                fulldisplay, hdebug, hlpfileno, hseqer, inhelp, nlmode,
                orgstid, qagain, qda, qnewstmt, qstoreblk, qsw, queryseq
            Constants
                endfil, entind, tppair
        %
    % Declarations %
        LOCAL
            helpad,      % work space %
            ptr REF;
        LOCAL STRING namstr[1000];
        LOCAL CONSTANT typename = 32;
```

```
    REF gda, qwa, gsw, qstorblk, qnewstmt;
  inhelp _ gagain := 0;     % gagain set ONLY here. %
  IF inhelp THEN
    BEGIN
    % We are in the help command already.  Show the help branch. %
    *namstr* _ "NINE,NLS,HELP";
    helpad _ $namstr;
    help2show(typename, $helpad, 0);
    DROP(ALL);
    RETURN;
    END;
  % open current tool's HELP file to initialize search %
    IF FIND SF(*namstr*) [",] THEN
      % See if the user explicitly specified a file %
      hlpfileno _ lcfile()
        % just initialize hlpfileno; the file will be loaded
        later (by the link follower) %
    ELSE % Look for <toolname>.NLS %
      IF NOT hlpfileno _ clohfil($tool) THEN
        % NLS is the default file %
        IF NOT hlpfileno _ clohfil($"NLS") THEN
          BEGIN
          *lit* _ "No help has been provided for ", *tool*;
          err($lit);
          END;
  % remember the file in which we started %
    helpfirstfile _ hlpfileno;
  INVOKE(elusivebug);
  % General initialization %
    &qda _ &qsw _ &qstorblk _ &qnewstmt _ &menusw _ 0;
    % Set flag saying we're in Help command. %
      inhelp _ TRUE;
    % Initialize global error flag %
      hseqer _ FALSE;
    % Save old da's (and jump stacks) and overlay a new one upon
    quitting, these will be restored %
      &qda _ pushda(1: &qwa);
    % The sequence generator for query/help. %
      qda.dafrzl _ 0;
      qda.davspec _ defvs1;
      qda.davspc2 _ defvs2;
      qda.dausgcod _ $queryseq;
      qda.davspec.vsusgf _ TRUE;
      qda.davspec.vsrind _ TRUE;                                  5A8E6
      qda.davspec.vsindf _ FALSE;
      qda.davspec.vspagf _ FALSE;
      qda.davspec.vsbrof _ TRUE;
      gdavspc _ qda.davspec;
      gdavs2  _ qda.davspc2;
    % Initialize pre-search pointers % % Get the stids of the top
    and the entry point--  this is not used %
      qda.dacsp _ orgstid;
      qda.daccnt _ 1;
      qda.dacsp.stfile _ hlpfileno;
    % Initialize the query stack areas. %
      qstrinit( orgstid, orgstid );
```

```
        curindex _ entind;
        confre _ 0;
        newstk _ TRUE;
    % Set up address of help command for this tool %
      IF &helplink THEN % context provided by user [and eventually
      CLI] %
        BEGIN
        &ptr _ ELEM #helplink#[tppair];      % tppair = 2 %
        *namstr* _ SF(ptr) SE(ptr);
        END
      ELSE namstr.L _ 0;  % null namstr %
    % Print out the first node. %
      helpad _ $namstr;
      help2show(typename, $helpad, 0);
    DROP(ALL);
    RETURN;
    (elusivebug) CATCHPHRASE ();                                    5A13
      CASE SIGNALTYPE OF
        = aborttype:
          BEGIN
          DROP(ALL);
          helpterm();
          err($"I am the elusive Help command bug:
          tell FEEDBACK details of what you were doing.");
          END;
        ENDCASE
          BEGIN
          helpterm();
          CONTINUE;
          END;
    END.

(helpterm)   % Terminates/cleans up the HELP system %
PROCEDURE;                                                          5B
    % Procedure description
      FUNCTION
        Terminates the help command.  Releases allocated storage
        (display area, window, string storage, etc.) allocated in
        the course of operating help.
      ARGUMENTS
        None.
      RESULTS
        None.
      NON-STANDARD CONTROL
        SIGNALS GENERATED
          None.
        SIGNALS CAUGHT
          All ABORT signals are terminated.
      GLOBALS
        Variables
          inhelp, qagain, menusw, qsw, qnewstmt, qstorblk, qda,
          qwa
        Constants
        Lists
      %
    % Declarations %
```

```
      REF qda, qwa, qsw, qstorblk, qnewstmt;
   IF qagain THEN RETURN; % Set TRUE if help called from in help:
   don't clear up %
   % Reset help flag %
      inhelp _ FALSE;
   IF &menusw THEN closeseq(&menusw := 0);
   IF &qsw THEN closeseq(&qsw := 0);
   IF &qnewstmt THEN freestring(&qnewstmt:=0, $dspblk);
   IF &qstorblk THEN freestring( &qstorblk:=0, $dspblk );
   IF &qda THEN popda(&qda, &qwa);
   RETURN;
   END.

(helpring)   % Steps back through previous views %
PROCEDURE (nwindex, values REF LIST % => BOOLEAN, INTEGER, INTEGER
%);                                                                  5C
   % Procedure description
      FUNCTION
         Provides feedback for stepping through the context ring
         with HELP's "BACK" command.
      ARGUMENTS
         nwindex - INTEGER - contains the ring index of the last
         statement that we asked the user if he wanted to see; 999
         if we are at the beginning:
      RESULTS
         1st - BOOLEAN - TRUE if there is a statement to ask the
         user about; FALSE if there are no more statements
         2nd - INTEGER - the ring index of the next statement to ask
         the user about; 0 if there are no more statements
         3rd - INTEGER - the stid of the next statement to ask the
         user about; 0 if there are no more statements
      NON-STANDARD CONTROL
         SIGNALS GENERATED
            err($"No text block associated with node")
               if HELP loses track of a previously displayed
               statement
      GLOBALS
         Variables
            conrng, curindex, entcon, rnglnt
         Constants
            entind, txttyp
      %
   % Declarations %
      LOCAL conad REF, stid, stdb, len;
      LOCAL TEXT POINTER tp1, tp2;
      LOCAL STRING temp[253;
      REF conrng;
   IF nwindex = 999 THEN nwindex _ curindex;
      % dav: Initialization to eliminate calls on qbkint. %
   % advance through the ring %
   % If we are at the beginning, say we can go no further. %
      IF nwindex = 0 THEN
         BEGIN
         dismes(2, $"No others have been shown");
         #values#[1] _ FALSE;
         #values#[2] _ curindex;
```

```
        #values#[3] _
           IF curindex = entind THEN entcon
           ELSE [&conrng + curindex*rnglnt];
        RETURN;
        END;
   &conad _
      IF nwindex = entind THEN $entcon
      ELSE &conrng + nwindex*rnglnt;
   IF nwindex # entind THEN
      BEGIN
      nwindex _ conbck(nwindex);
      &conad _
         IF nwindex = entind THEN $entcon
         ELSE &conrng + nwindex*rnglnt;
      END;
   stid _ conad;
   % display the current statement %
      % display first 20 chars of stmt in name area %
         % get statement length %
            IF NOT lodprop( stid, txttyp : stdb) THEN
               err($"No text block associated with node");
            len _ [stdb].schars + 1;          % number of chars %
         % construct text ptrs to each end of stmt %
            tp1 _ stid;
            tp1[1] _ 1;
            IF NOT FIND SF(tp1) [EOL ^tp2 _tp2 /  "##" ^tp2 _2tp2 ]
            THEN
               BEGIN
               tp2 _ stid;
               tp2[1] _ len;
               END;
            % Truncate %
               tp2[1] _ MIN(20, tp2[1]);
         % assign something to the string "temp" %
            IF tp2[1] > 1 THEN *temp* _ tp1 tp2, "... "
            ELSE *temp* _ "<NULL> ";
      dismes(1, $temp);
   % save the current state in the result record %
      #values#[1] _ TRUE;
      #values#[2] _ nwindex;
      #values#[3] _ stid;
   RETURN;
   END.

(helpshow)   % Dummy routine for CML/L10 communication %
PROCEDURE (type REF LIST, param REF LIST, stid);                    5D
   % Procedure description
      FUNCTION
         Dummy routine for communicating between the CML and
         help2show.  Necessary because help2show wants to be called
         from both the front end and the back end.
      ARGUMENTS
         type - LIST - a CML command word indicating the type of
         showing to be done: up, name, next, back, menu
            Example: ([INTEGER 32]  [LITERAL name]  [INTEGER 1])
         param - LIST - additional information depending on the
```

```
        value of 'type'
           Example: ([INTEGER 30]  [LITERAL foo]  [INTEGER 1])
        stid - INTEGER - the stid of the back node when 'type' =
        'back'
     RESULTS
        none
     NON-STANDARD CONTROL
        none
     GLOBALS
        Variables
           none
        Constants
           cwtype, tppair
     %
  % Declarations %
     LOCAL helpad, ptr REF;
     LOCAL STRING s[1000];
     LOCAL CONSTANT typename = 32;
  IF ELEM #type#[cwtype] = typename THEN     % cwtype = 1 %
     BEGIN
     &ptr _ ELEM #param#[tppair];            % tppair = 2 %
     *s* _ SF(ptr) SE(ptr);
     helpad _ $s;
     help2show(typename, $helpad, 0);
     END
  ELSE help2show(ELEM #type#[cwtype], $param, stid);
  RETURN;
  END.

(help2show)   % Prints a statement and its menu %
PROCEDURE (type, param REF, stid % => none %);                    5E
  % Procedure description
     FUNCTION
        Does the actual printing of a statement and its menu.  Sets
        global moremen FALSE if the menu was finished; TRUE if more
        is to be displayed.
     ARGUMENTS
        type - INTEGER - indicates the type of showing to be done:
        up, name, next, back, menu
        param - ADDRESS - additional information depending on the
        value of 'type'; usually the address of a block containing
        a string
           Example: ["foo"]
        stid - INTEGER - the stid of the back node when 'type' =
        'back'
     RESULTS
        none
     NON-STANDARD CONTROL
        SIGNALS GENERATED
           err($"Help system error:
           Invalid parameter passed to helpshow")
              if 'type' was not one of: up, name, next, back, menu
     GLOBALS
        Variables
           curindex, qda, qsw
        Constants
```

```
            none
        %
    % Declarations %
        LOCAL STRING emptmes[300];
        LOCAL CONSTANT
            typename = 32, typeup = 54, typeback = 59, typenext = 61,
            typemenu = 71;
        REF qda, qsw;
    CASE type OF
        = typename:
            % param contains a pointer to a string with either a menu
            number or a word to be used in a name search %
            IF qsearch(param.stpsid, curindex : stid, curindex) THEN
                % param.stpsid is the string address %
                BEGIN
                conint(stid);
                chkdsp(stid);
                END
            ELSE
                BEGIN
                *emptmes* _ *[param.stpsid]*, "?";
                dismes(1,$emptmes);
                END;
        = typenext:
            % show the rest of the menu that would not fit %
            IF &qsw THEN chkdsp(0)
                % Call qdisp in continue mode. %
            ELSE dismes(2, $"Last menu was complete: No more");
        = typeback:
            % param is the ring index of the back node to be displayed;
            stid contains the actual stid %
            BEGIN
            curindex _ &param;
            chkdsp(stid);
            END;
        = typeup:
            % find the source of the current node %
            BEGIN
            stid _ qda.dacsp _ qgetup(qda.dacsp, &qda, $emptmes);
            conint(stid);                   % update context stack %
            chkdsp(stid);                   % display node %
            END;
        = typemenu:
            % param contains the stid of a menu item %
            dismes(1, $"notyet");
        ENDCASE
            err($"Help system error:
            Invalid parameter passed to helpshow");
    RETURN;
    END.

% ******* "chelp" core routines ******** %
% Initialization and display %
    (chkdsp) PROCEDURE (stid);                                    7A
        % displays node at stid %
            CASE qdisp(stid) OF
```

```
        = 1: % Everything OK; no more to be shown. %
           BEGIN
           IF hseqer THEN err($"Ran out of space making menus.");
           moremen _ FALSE;
           RETURN
           END;
        = -1: % Things OK for now, but must get more interaction
        from user. %
           BEGIN
           IF hseqer THEN err($"Ran out of space making menus.");
           moremen _ TRUE;
           RETURN
           END;
        ENDCASE % Terrible error. %
           BEGIN
           moremen _ FALSE;
           err($"Data base portrayal trouble.  Call ARC!");
           END;
     RETURN;
     END.

(clohfil) PROCEDURE (astring);                                          7B
   % Given a tool name, returns a jfn for the help file for that
   tool.  Returns false if there is no help for that tool. %
   LOCAL STRING dirstr[40], hfilnm[100];
   REF astring;
   LOCAL filenumber;
   *dirstr* _ "<USERGUIDES>";
   (try1):                                                              7B6
      INVOKE(didntwork, try2);
      *hfilnm* _ *dirstr*, *astring*, ".NLS;" ;
      filenumber _ cloafil($hfilnm);
      DROP(ALL);
      RETURN(filenumber);
   (try2):                                                              7B7
      INVOKE(didntwork, try3);
      *hfilnm* _ "<XPROGRAMS>", *astring*, ".NLS;" ;
      filenumber _ cloafil($hfilnm);
      DROP(ALL);
      RETURN(filenumber);
   (try3):                                                              7B8
      DROP(ALL);
      RETURN(0);
   END.

(conint) PROCEDURE (stid);                                              7C
   REF conrng, curstk, qda;
   % Initializes context %
      newstk _ TRUE;
      IF NOT conrng[confre*rnglnt] THEN
         % found a free ring element; bump index and confre %
         BEGIN
         curindex _ confre;
         IF confre >= rngmax THEN confre _ 0
         ELSE BUMP confre;
         END
```

```
      ELSE % no empty ring elements so reuse the oldest %
         BEGIN
         curindex _ confre;
         freemenu(&conrng + curindex*rnglnt);
         % go free up the stack we are going to use now %
         IF confre >= rngmax THEN confre _ 0
         ELSE BUMP confre;
         END;
      &curstk _ &conrng + curindex*rnglnt;
      curstk _ stid;
      qda.dacsp _ stid;
         % so that includes will have proper file for link parses %
      qda.daccnt _ 1;
      hlpfileno _ stid.stfile;
      % mark the current file not to be closed. %
         [fintadr(hlpfileno)].flnoclos _ TRUE;
      RETURN;
      END.

(freemenu) PROCEDURE (conptr);                                          7D
   % given the address of a context node this procedure frees up any
   menus associated with this node%
   LOCAL menblk, nxtmenblk;
   REF conptr,menblk,nxtmenblk;
      IF conptr.stktyp THEN &menblk _ conptr.constk
      ELSE RETURN(TRUE);
      conptr.stktyp _ conptr.constk _ conptr.totcnt _ 0;
      LOOP
         BEGIN
         &nxtmenblk _ menblk.conlnk;
         IF NOT freeblk(&menblk-2, menblk[-1]) THEN err($"attempt to
         free invalid storage block");
         IF &nxtmenblk = -1 THEN RETURN(TRUE) ELSE &menblk _
         &nxtmenblk;
         END;
      END.
(moreterm) PROCEDURE;                                                   7E
   %  Must be at a lower level because of symbol conflicts. Closes
   sequences and resets viewspecs. %
      % Clean up the last work area if there is one %
      seggen(&qsw);                        % To clean up %
      qda.davspec _ qsw.swvspec;
      qda.davspc2 _ qsw.swvsp2;
      IF &menusw THEN closeseg (&menusw:=0);
      IF &qsw THEN closeseg (&qsw:=0);
      RETURN;
      END.

(qdisp) PROCEDURE (stid);                                               7F
   % (dav) Displays a statement and its menu %
      LOCAL dummy, endflg;
      REF qsw, qda, menusw;
      % set up the user sequence generator address for use by
      printg.  The only case in which qsw is non-zero here is on
      continued menus %
         IF NOT &qsw THEN
```

```
            BEGIN
            IF NOT stid THEN RETURN(FALSE);
            % A new sequence %
               &qsw _ openseq(stid, stid, qda.davspec, qda.davspc2,
               qda.dausgcod, qda.dacacode);
            % Initialize globals for queryseq %
               qspcreset(TRUE); % default query viewspecs %
               qmenucnt _ 0;
            qda.dacsp _ stid;
            qda.daccnt _ 1;
            overscreen _ FALSE;
            END
         ELSE IF NOT moremenu THEN
            BEGIN
               % This is an error condition; if there is a sequence
               around, should not get in here unless moremenu is
               TRUE -- more to be shown.  Close the sequence and
               return FALSE.  Clean up the last work area if there
               is one %
            seggen(&qsw); % To clean up %
            qda.davspec _ qsw.swvspec;
            qda.davspc2 _ qsw.swvsp2;
            IF &menusw THEN closeseq( &menusw := 0);
            IF &qsw THEN closeseq(&qsw := 0);
            RETURN( FALSE );
            END;
      INVOKE(didntwork, label1);
      % Display node and menu. %
         IF nlmode = fulldisplay THEN
            dafrmt(&qda, &qsw : dummy, endflg)
         ELSE
            BEGIN
            cprint(&qda, stid, stid, brnchv, FALSE, &qsw);
            endflg _ TRUE;
            END;
      % Check if finished or if end of screen or menu limit reached.
      %
         IF qsw.swcstid = endfil          % printed everything %
         OR (&menusw AND
            getftl(menusw.swcstid))       % printed tail of menu %
         OR inptrf THEN                   % <CTRL-O> %
            BEGIN
            IF &menusw THEN closeseq( &menusw:=0);
            IF &qsw THEN closeseq( &qsw:=0);
            qda.davspec _ qdavspc;
            qda.davspc2 _ qdavs2;
            IF hseger THEN
               err($"Ran out of space making menus.");
            DROP(ALL);
            RETURN(1);
            END
         ELSE
            BEGIN
            % Menu limit has been reached or end of screen hit %
               overscreen _ NOT endflg;
            % so we can go through again %
```

```
            qda.dacsp _ qsw.swstid _ qsw.swcstid;
         % user should be asked whether more desired, get answer
         %
            DROP(ALL);
            RETURN(-1);
         END;
      (label1):                                                    7F1G
         DROP(ALL);
         RETURN(FALSE);
      END.

(qstrinit) PROCEDURE (entstd, topstd);                             7G
   % qstrinit should allocate a large block from dspblk, put its
   address in qstorblk; this will be used to get smaller blocks.
   xhquit will deallocate this large block if it is non-zero. %
   % Called to initialize the stack area as a storage zone for
   help/query. Makes use of stgmgt routines to set up blocks in the
   display area block pool. Also responsible for initializing some
   global cells as well as the context ring %
      % Allocate string %
         &qnewstmt _ getstring( 2000, $dspblk);
      % Get context ring and initialize. %
         IF NOT (&qstorblk _ getblk(1777B, $dspblk)) THEN
            err($"System storage problem. Call ARC programmer");
         makezone(&qstorblk, 70B, 70B, 1777B);
         IF NOT (&conrng _ getblk(67B, &qstorblk)) THEN
            err($"System storage problem. Call ARC programmer");
         conrng[1] _ &qstorblk; % first word in blocks is the
         address of the zone from which it came. %
         &conrng _ &conrng + 2; % conrng now points beyond the block
         header and the zone address to the actual stack. %
         confre _ 0;
      % Initialize the top and entry point. %
         % For the top point, perhaps we should have two stids: stid
         of the "introduction" branch for the data base and stid of
         the "directory" beanch. Will not need (should not have?) a
         context stack for the top! Coming into this procedure,
         topstd should be the stid of the origin of the data base.
         From this node, we may extract links to the "intro" and
         directory branches (which may be the same if desired.)
         CHANGE THIS CODE! %
            topcon _ topstd; % first word is stid %
            topcon[1] _ 0; % totcnt, constk, and stkyp fields are 0
            %
         entcon _ entstd; % first word is stid %
         entcon[1] _ 0; % totcnt, constk, and stkyp fields are 0 %
      % Initialize global for building entry menu %
         % The first call on pushent should get a stack if necessary
         nd put the type and address ithe context ring. The free
         pointer should be incremented then. %
         $curstk _ $entcon;
         newstk _ 1;
      RETURN; % and then print the entry menu for help %
      END.

% Search control %
```

```
(conbck) PROCEDURE (index);                                          8A
   % back up one context if possible. otherwise go to etrypoint %
   % Check for entrypoint BEFORE calling this procedure !!! %
      CASE index OF
         >0: IF (index _ index-1) = confre THEN index _ entind;
         ENDCASE % =0 %
            index _ IF conrng[rngmax*2] THEN rngmax ELSE entind;
      RETURN( index );
      END.

(menustart) PROCEDURE (mennum, adr);                                 8B
   % Accepts address of a string to be updated and one to be
   evaluated.  Updates the first string to contain just the menu
   number if the second begins with a menu number.  Updates the
   second string with whatever is after the menu number or NULL if
   only non printing characters.  Returns false if it does not begin
   with a menu number. %
      LOCAL TEXT POINTER ptr1, ptr2;
      REF adr, mennum;
      mennum.L _ 0;
      IF NOT ( FIND SF(*adr*) ^ptr1 $NP D $D ^ptr2 (NP/ENDCHR) ) OR
      (FIND ptr1 $NP "0) THEN RETURN(FALSE);
      *mennum* _ ptr1 ptr2;
      *adr* _ ptr2 SF(*adr*);
      IF FIND SF(*adr*) $NP ENDCHR THEN adr.L _ 0;
      RETURN(TRUE);
      END.

(numfst) PROCEDURE (conad, stid, number, index);                     8C
   % The first item in the list is a menu number.  Count down
   through menu stack "n" items.  It is an error if the number is
   too high %
      LOCAL wrkaddr, nwindex;
      LOCAL STRING message[100];
      REF conad, wrkaddr;
      &conad _ gvalid(&conad, index: nwindex);
      IF conad.totcnt < number THEN
         BEGIN
         *message* _ STRING(number)," is an invalid menu number";
         dismes(2,$message); % for the display, we should perhaps
         put message on the screen %
         RETURN (FALSE,stid, index);
         END;
      &wrkaddr _ conad.constk;
      LOOP % over menu blocks in this stack %
         BEGIN
         IF number <= stkmax THEN
            BEGIN
            stid _ wrkaddr[number*2];
            RETURN (TRUE,stid, nwindex);
            END
         ELSE
            BEGIN
            &wrkaddr _ wrkaddr;
            number _ number - stkmax;
            END;
```

```
        END;
    END.

(qgetup) PROCEDURE (stid, da, vspstg);                                  8D
  % given a stid, display area, and viewspec string, finds the
  source of the current branch.  If it is the origin of a file,
  looks for "source: <link>" to define the source.  Puts viewspecs
  from the link in vspstg. %
    LOCAL STRING lnk[100];
    LOCAL TEXT POINTER ptr1, ptr2, mkr;
    REF vspstg, da;
    IF ( stid.stpsid = origin ) AND (FIND SF(stid) ["source: "]
    ^ptr1 "<  ["">] ^ptr2) THEN
      BEGIN
      *lnk* _ ptr1 ptr2;
      wfndit($ptr1, &vspstg, $mkr, &da);
      RETURN(mkr);
      END;
    RETURN(getup(stid));
    END.

(qsearch) PROCEDURE (list, index);                                      8E
  % Given the address of a string containing the node list of an
  item to be found and an index to the context ring element from
  which the search will start, search for the specified node.
  Returns TRUE and the stid of the found node (or FALSE and -1 in
  stid if nothing found). %
    LOCAL
      conad, % address of current context %
      stid, % of found item or -1 %
      nwindex, % of ring or old index if not found %
      number;
    LOCAL TEXT POINTER ptr, cursptr;
    LOCAL STRING vspstg[100], menum[100]; % String to be used for
    getting menu number %
    REF list, conad;
    IF NOT list.L THEN RETURN(TRUE, qda.dacsp, index);
    % Initialize for search %
      stid _ -1;
      nwindex _ index;
      vspstg.L _ fwrd _ 0;  % for bugging, not used in Help %
      &conad _
        IF index = entind THEN $entcon
        ELSE &conrng + index*rngint;
      glblstd _ 0;
    % see if it starts with a menu number, otherwise call wfndit %
      IF NOT menustart($menum,&list) THEN
        BEGIN
        cursptr _ qda.dacsp;  cursptr[1] _ 1;
        *list* _ "<, *list*, ">;  % so lnkprs can be used %
        FIND SE(*list*) CH ^ptr;
        IF NOT wfndit($ptr,$vspstg,&list,$cursptr,&qda) THEN
          BEGIN
          IF cursptr = qda.dacsp THEN % we assume the whole
          link is bad %
          RETURN(FALSE, -1, index);
```

```
            % *list* now contains the portion of the address that
              failed but we don't tell the user what failed ? %
              END;
           RETURN(TRUE, cursptr, index);
           END
        ELSE
           BEGIN
           number _ VALUE($menum);
           IF NOT numfst(&conad, stid, number, nwindex: stid,
           nwindex) THEN RETURN (FALSE, -1, nwindex);
           cursptr _ stid;  cursptr[1] _ 1;
           wbrsrh(&list,$cursptr,&qda);
           RETURN(TRUE,cursptr,nwindex);
           END;
     END.

(qvalid) PROCEDURE (conad, index);                                    8F
   % Check if this stack has any entries; back up if not %
     REF conad;
     UNTIL conad.corstk OR index = entind DO
        BEGIN
        index _ conbck( index );
        &conad _
           IF index = entind THEN $entcon
           ELSE &conrng + index*rnglnt;
        END;
     RETURN(&conad, index );
     END.

(samename) PROCEDURE (stmtnm, cursptr);                               8G
   % Compares a statement name with the one in the statement at
   cursptr.  Returns true if same else false. %
     LOCAL spcnmhash;
     LOCAL STRING locstr[100];
     REF stmtnm, cursptr;
     astruc(&stmtnm);
     spcnmhash _ hash(&stmtnm);
     IF NOT getnmf(cursptr) OR getnam(cursptr) # spcnmhash THEN
        RETURN(FALSE);
     CCPOS SF(cursptr);
     xtrnam($locstr, $swork, -1, 0);
     RETURN( IF *stmtnm* = *locstr* THEN TRUE ELSE FALSE ) ;
     END.

(samevw) PROCEDURE (vspstg, wda);                                     8H
   % returns true if the viewspecs evaluated in vspstg are the same
   as those in wda false otherwise %
     LOCAL same, vs1, vs2, cacode, usedgen;
     REF vspstg, wda;
     *vspstg* _ *defvspc*, *vspstg*;
     % save viewspecs %
        vs1 _ wda.davspec; vs2 _ wda.davspc2;
        cacode _ wda.dacacode; usedgen _ wda.dausgcod;
     IF vspstg.L # 0 THEN % set up viewspecs %
        BEGIN
        % if you typed in the name and then hit CA, there will be
```

```
        no viewspecs, therefore, no matter what the view, you will
        go to a higher level search.  The reasoning is if you
        wanted a map view of the current location, you would not
        have typed the name, you wuld have just hit CA %
        feedit(&wda,&vspstg); % evaluate the viewspecs %
        END;
      IF (( cspvs # wda.davspec ) OR ( cspvs[1] # wda.davspc2 ))
      THEN same _ TRUE ELSE same _ FALSE ;
      % replace old viewspecs to fool howformat %
        wda.davspec _ vs1; wda.davspc2 _ vs2;
        wda.dacacode _ cacode; wda.dausqcod _ usedgen;
      RETURN (same) ;
      END.

(srchfndx) PROCEDURE (cursptr, movmkr, stmtnm, wda);                    8I
   % follows index links to other files %
      LOCAL hostno, ndxcnt;
      LOCAL TEXT POINTER savmkr, ptr1, ptr2;
      LOCAL STRING usrnam[40], filenm[80], adr[100], vspcs[100];
      LOCAL ldstr[35];
      REF wda, cursptr, movmkr, stmtnm;
      IF inptrf THEN RETURN(FALSE);  % the user typed ctrl-0 %
      ndxcnt _ 0;
      movmkr     _ savmkr     _ cursptr;
      movmkr[1] _ savmkr[1] _ 1;        % changed to endfile %
      LOOP % see if this file has it's own index %
         BEGIN
         % Look for an index denoted by the string "Index in <...>"
         %
            movmkr.stpsid _ origin;
            IF NOT FIND SF(movmkr) [^"ndex in <"] ^ptr1 _ptr1 [^'>]
            ^ptr2 THEN
               BEGIN
               % Look in statement 1 %
                  movmkr _ getsub(movmkr);
                  IF NOT FIND SF(movmkr) [^"ndex in <"] ^ptr1 _ptr1
                  [^'>] ^ptr2 THEN
                     EXIT LOOP;
               END;
         dismes(1,$"searching index");
         % find the file (viewspecs are ignored) %
            INVOKE(didntwork, badindex);
            caddexp($ptr1, $ptr2, &wda, $movmkr);
            DROP(ALL);
            IF movmkr.stfile = savmkr.stfile THEN EXIT LOOP;
               % we've aready looked here %
         % find the name %
            savmkr _ movmkr;
            IF inptrf THEN RETURN(FALSE);  % the user typed ctrl-0 %
            *adr* _ *stmtnm*;  % lookup says it modifies string %
            movmkr _ lookup($movmkr, $stmtnm, nametyp);
            *stmtnm* _ *adr*;  % lookup says it modifies string %
            IF movmkr # endfil THEN
               BEGIN
               cursptr _ movmkr; cursptr[1] _ 1;
               RETURN(TRUE);
```

```
            END;
         IF inptrf THEN RETURN(FALSE);  % the user typed ctrl-0 %
      movmkr _ savmkr;
      BUMP ndxcnt;
      IF ndxcnt > 10 THEN
         BEGIN
         dismes(1, $"reached limit of 10 index files");
         EXIT LOOP;
         END;
      END;  % repeat loop %
   RETURN(FALSE);
   (badindex):                                                    8I1L
      DROP(ALL);
      IF NOT inptrf THEN dismes(1,$"Bad index link");
      RETURN(FALSE);
   END.

(wbranm) PROCEDURE (stmtnm, cursptr, wda);   % <,:b> %              8J
   % If not in branch, look for a link.  If the link is not
   completely accurate, forget it.  To search a link in every
   statement in substructure is potentially too time consuming. %
      REF stmtnm, cursptr, wda;
      LOCAL cnnt;
      LOCAL STRING wadrsave[150], vspstg[100];
      LOCAL TEXT POINTER movmkr, ptr1, ptr2;
      cnnt _ 0;
      LOOP
         BEGIN
         IF inptrf THEN RETURN(FALSE);        % user typed ctrl-0 %
         movmkr _ cursptr; movmkr[1] _ 1;
         movmkr _ lookup($movmkr, $stmtnm,
            (IF cursptr.stpsid = origin THEN nametype ELSE
            braname));
         IF movmkr # endfil THEN              % we found it %
            BEGIN
            cursptr _ movmkr;
            RETURN(TRUE);
            END;
         % see if there is a link to somewhere else %
            IF inptrf THEN RETURN(FALSE);     % user typed ctrl-0 %
            movmkr _ glblstd _ cursptr;
            IF NOT (FIND SF(movmkr) [">] ^ptr2 _ptr2 < ["<] ^ptr1
               ("##"/EOL/CR)) OR (FIND ptr1 > CH (",/) $NP (":/">))
               THEN RETURN(FALSE);
         % take it %
            *wadrsave* _ ptr1 ptr2, ">;
            IF NOT wfndit($ptr2,$vspstg,$wadrsave,&cursptr,&wda)
            THEN RETURN(FALSE);
         IF cursptr = movmkr THEN RETURN(FALSE);
         movmkr _ cursptr;
         IF cnnt > 10 THEN
            BEGIN
            <err>($"looping link?  ");
            RETURN(FALSE);
            END;
         BUMP cnnt;
```

```
        END;
    END.

(wbrsrh) PROCEDURE (wadr, cursptr, wda);     % <,:b> %                    8K
  % ? %
    LOCAL TEXT POINTER ptl1, wnmptr, movmkr, ptr1, ptr2;
    LOCAL STRING stmtnm[100], vspstg[100];
    LOCAL msg REF;
    REF wda, wadr, cursptr;
    FIND SF(*wadr*) ^ptr2;
    LOOP
      % <,:b> to take the user as far as we can after the first
      name %
      BEGIN
      IF FIND ptr2 > $NP ENDCHR THEN RETURN(TRUE);
      movmkr _ cursptr; movmkr[1] _ 1;
      IF FIND ptr2 > SP $NP (L/"@) ^ptr1 _ptr1 $(LD/"@/""/"-)
      ^ptr2 THEN
        % there is a stacked name, %
        BEGIN
        *stmtnm* _ ptr1 ptr2;
        IF NOT wbranm($stmtnm, $movmkr, &wda) THEN
          BEGIN
          IF getsub(cursptr) # cursptr THEN
            *wadr* _ *stmtnm*, "?  You can pick one and try
            again"
          ELSE *wadr* _ *stmtnm*;
          RETURN(FALSE);
          END;
        cursptr _ movmkr;
        IF inptrf THEN
          BEGIN
          *wadr* _ *stmtnm*;
          RETURN(FALSE);          % the user typed ctrl-0 %
          END;
        END
      ELSE
        BEGIN
          % set up content searches, link searches, and branch
          searches for caddexp (change caddexp to do this
          stuff) %
        IF FIND ptr2 > $NP ENDCHR THEN RETURN(TRUE);
        IF FIND ptr2 > $NP '" ^ptr1 THEN
          BEGIN
          IF FIND ptr1 > [""] (*=/) $LD ^ptr2 _ptr1 THEN
            BEGIN
            % *******
              ON SIGNAL
                = gaderr:
                  BEGIN
                  IF [sysmsg].L # 0 THEN *wadr* _
                  *[sysmsg]*;
                  RETURN(FALSE);
                  END;
                ELSE; ******* %
            INVOKE(syserror, wbrsrer);
```

```
                caddexp($ptr1, $ptr2, &wda, $movmkr);
                % ******* ON SIGNAL ELSE; ******* %
                DROP(ALL);
                IF movmkr # endfile THEN cursptr _ movmkr;
                cursptr[1] _ 1;
                END
             ELSE *wadr* _ ptr2 SE(*wadr*), '"; % add the quote %
             REPEAT LOOP;
             END;
          FIND0 ptr2 ^wnmptr;
          LOOP
             IF FIND wnmptr > [SP] ^ptr1 _ptr1 (L/'@) ^ptl1 _ptl1
             $(LD/'-/''/'@) ^wnmptr THEN
                ST ptr1 ptl1 _ " !"
             ELSE
                BEGIN
                FIND ptr2 ^wnmptr;
                EXIT LOOP;
                END;
          LOOP
             IF FIND wnmptr > [D] ^ptr1 _ptr1 $D ^ptl1 '. ^wnmptr
             (ENDCHR/SP) THEN
                ST ptr1 wnmptr _ '", '<, '", '=, ptr1 ptl1, "cls"
             ELSE EXIT LOOP;
          FIND ptr2 $NP ^ptr1 SE(*wadr*) $NP ^ptr2;
          IF ptr1[1] < ptr2[1] THEN          % call caddexp %
             BEGIN
             % *******
                ON SIGNAL = gaderr: GOTO wcaderr; ELSE; ******* %
             INVOKE(syserror, wbrsrer);
             caddexp($ptr1, $ptr2, &wda, $movmkr);
             IF movmkr # endfile THEN
                BEGIN
                cursptr _ movmkr;
                cursptr[1] _ movmkr[1];
                END;
             END;
          DROP(ALL);
          RETURN(TRUE);
          (wbrsrer):                                          8K1F6K
             DROP(ALL);
             RETURN(FALSE);
          END;
       END;          % repeat loop %
    % *******
       (wcaderr):                                             8K1G1
          IF [sysmsg].L # 0 THEN *wadr* _ *[sysmsg]*;
          RETURN(FALSE); ******* %
    (syserror) CATCHPHRASE (:&msg);                          8K1H
       CASE SIGNALTYPE OF
          = aborttype:
             BEGIN
             IF SIGNAL = gaderr THEN
                BEGIN
                IF [msg].L # 0 THEN *wadr* _ *[msg]*;
                END;
```

```
            TERMINATE;
            END;
         ENDCASE CONTINUE;
   END.

(wIndit) PROCEDURE (ptr, vspstg, wadr, cursptr, wda);                    8L
   % Given an address to a pointer to a wuc link, an address to a
   viewspec string, an address to the string containing the wuc link
   (wadr), and a pointer to be updated with the new location in the
   file, evaluates the link in relation to its current location,
   adds viewspecs to vspstg and returns true if the address was
   completely successful.  Goes as far as it can and updates
   cursptr, returns false and the portion of the link that failed in
   wadr if it failed. %
      LOCAL STRING filenm[100];
      LOCAL msg REF, x;
      REF wda, ptr, vspstg, wadr, cursptr;
      % *******
         ON SIGNAL ELSE
            BEGIN
            IF NOT [sysmsg].L THEN *[sysmsg]* _ "huh";
            *wadr* _ *[sysmsg]*;
            RETURN (FALSE);
            END; ******* %
      (wferror) CATCHPHRASE (:&msg);                                     8L1F
         CASE SIGNALTYPE OF
            = aborttype:
               BEGIN
               IF NOT [msg].L THEN *[msg]* _ "huh";
               *wadr* _ *[msg]*;
               TERMINATE;
               END;
            ENDCASE CONTINUE;
      INVOKE(wferror, confused);
      x _
         IF <wprsfile>(&ptr, &vspstg, &wadr, $filenm) THEN
            <wgetfile>($filenm, &wadr, &cursptr, &wda)
               % we don't call wgetadr if a file has been specified
               because since wgetadr continues searching higher
               levels if the first name is not found it would be
               inconsistant with the rule that succeeding elements
               are always taken in relation to the preceeding
               elements. %
         ELSE IF wadr.L THEN
            <wgetadr>($wadr, $cursptr, &wda, &vspstg)
         ELSE TRUE;
      DROP(ALL);
      RETURN(x);
      (confused):                                                        8L1J
         DROP(ALL);
         RETURN(FALSE);
      END.

(wgetadr) PROCEDURE (wadr, cursptr, wda, vspstg);                        8N
   % Finds an address within a multi-file database %
      LOCAL onlyWrd;
```

```
    LOCAL TEXT POINTER ptr, ptr1, ptr2;
    LOCAL STRING stmtnm[120];
    REF wadr, cursptr, wda, vspstg;
    onlywrd _ FALSE;
    IF fwrd OR (FIND SF(*wadr*) (L/"@/ % "&/ % "(/")) ^ptr)  THEN
       BEGIN
          % starts with statement name not preceeded by space.  If
          you specify the same name as where you already are, your
          current branch will not be searched unless you place a
          space in front of the name.  This limits the search to
          your current branch or a link in your current branch. %
       nmdr($ptr,$ptr1,$ptr2);      % Take out name %
       *stmtnm* _ ptr1 ptr2;
       IF FIND ptr2 > $NP ENDCHR THEN onlywrd _ TRUE
       ELSE
          BEGIN
          onlywrd _ FALSE;
          *wadr* _ ptr2 SE(*wadr*);
          END;
       IF NOT <wstrtnm>($stmtnm, &cursptr, &wda, &vspstg, onlywrd)
          THEN BEGIN
          IF cursptr.stfile = helpfirstfile THEN
             BEGIN    % Already tried the initial file %
             *wadr* _ *stmtnm*;
             RETURN(FALSE);
             END;
          FIND cursptr ^ptr;    % save cursptr %
          cursptr.stfile _ helpfirstfile;
          cursptr.stpsid _ origin;
             % Start at the origin of the initial file %
          IF NOT <wstrtnm>($stmtnm, &cursptr, &wda, &vspstg,
             onlywrd) THEN
                % This may search some (index) files that have
                already been searched.  Oh well. %
             BEGIN
             FIND ptr ^cursptr;     % restore cursptr %
             *wadr* _ *stmtnm*;
             RETURN(FALSE);
             END;
          END;
       END;
    IF NOT onlywrd THEN              % get the rest of the address %
       RETURN(<wbrsrh>(&wadr,&cursptr,&wda))
    ELSE RETURN(TRUE);
    END.

(wgetfile) PROCEDURE (filenm, wadr, cursptr, wda);                  8N
   % gets file.  If file not on line, signals with FILE NOT ON LINE
   message.  Finds location in file if any, sets up default
   viewspecs, search specs and returns TRUE.  If it fails, puts bad
   name in wadr. %
      LOCAL TEXT POINTER ptr1, ptr2, movmkr;
      LOCAL STRING locadr[100], dumbvs[100];
      REF wda, wadr, cursptr;
      *locadr* _ "0";
      cursptr _ fstid(filenm);
```

```
      % set up default search type for file (when implemented) %
      % set up default viewspecs for file %
         IF FIND SF(cursptr) ["default view: <:"] ^ptr1 $LD "> ^ptr2
         _ptr2 THEN *defvspc* _ ptr1 ptr2;
      cursptr[1] _ 1;
      IF wadr.L THEN RETURN(wbrsrh(&wadr,$cursptr,&wda));
      RETURN(TRUE);
      END.

(wprsfile) PROCEDURE (ptr, vspstg, wadr, filenm);                    8C
   % Parses the link in wadr and returns TRUE if there is a file
   address %
      % declarations %
         LOCAL TEXT POINTER ptr1, ptr2;
         LOCAL ndxcnt, hostno;
         LOCAL STRING wstovs[100], wstoad[100];
         LOCAL STRING hnmono[5], locflnm[40], usrnam[40];
         LOCAL ldstr[35];  % link parsing record %
      REF ptr, vspstg, wadr, filenm;
      FIND SE(*wadr*) ^ptr1;
      lnkprs($ptr1, $ldstr);
      ldstr[lfn] _ ptr.stfile; % because we copied bugged link into
      wadr and called lnkprs with pointer to copied link (to allow
      first word of link to be named).  Why do typed-in links where
      ptr has no stfile work? %
      CASE hostno _ lnkpspc( 1, 0, $usrnam, $filenm, $wstoad,
      $wstovs, $ldstr) OF
         = lhostn: NULL;  % local host %
         ENDCASE
            BEGIN
            *wadr* _ "Remote file access not implemented";
            RETURN(FALSE);
            END;
      *wadr* _ *wstoad*;
      *vspstg* _ *vspstg*, *wstovs*;
      IF ldstr[fe+1] > ldstr[fs+1] THEN RETURN (TRUE);  % file name
      %
      RETURN (FALSE);
      END.

(wstrtnm) PROCEDURE (stmtnm, cursptr, wda, vspstg, onlywrd);         8P
   % finds the first name in a multi-file database and returns true.
    Returns false if it can't find it.  Uses vspstg to check for
   view if necessary when the same name has been requested twice in
   a row. %
      % declarations and REFs %
         LOCAL TEXT POINTER movmkr, ptr1, ptr2;
         LOCAL STRING savenm[100];
         REF vspstg, stmtnm, cursptr, wda;
      % initializations %
         movmkr _ cursptr;
         movmkr[1] _ cursptr[1];
      IF cursptr # glblstd THEN
         % we have not been called from wbranm so do the branch
         search %
         BEGIN
```

```
    IF fwrd OR <samename>($stmtnm, &cursptr) THEN
       % Check if the view is the same.  If the first word of a
       statement was bugged or the current statement's name has
       been requested, then we assume the user wants a more
       general concept at a higher level unless a different
       view has been specified. %
       BEGIN
       IF (cursptr = wda.dacsp) AND (wda.daccnt = 1) THEN
          % Check if we are at the same statement %
          BEGIN
          IF NOT ( inhelp %OR athelp% ) AND onlywrd AND NOT
          <samevw>(&vspstg, &wda) THEN RETURN(TRUE);
             % If there were something following the name (NOT
             onlywrd), then since in every case you can find
             the address in relation to the current location
             without typing the name of the current location we
             assume you were looking for a statement with the
             same name elsewhere and so skip this. %
          END
       ELSE IF fwrd THEN RETURN(TRUE);
          % take the user there even if it's not named %
          % if fwrd and not onlywrd, then we don't go searching
          higher level %
       END
    ELSE        % search current branch %
       BEGIN
       *savenm* _ *stmtnm*;
       IF <wbranm>($stmtnm, $movmkr, &wda) THEN
          BEGIN
          cursptr _ movmkr;
          RETURN(TRUE);
          END;
       IF inptrf THEN RETURN(FALSE);
       movmkr _ cursptr;  movmkr[1] _ 1;  % movmkr = endfil %
       END;
    END
ELSE glblstd _ 0;
% Get search type %
    % Before changing the search type to alphabetic in the top
    index, this should be changed to use the search type
    defined in the origin statement.  This might be the top
    index.  We should either set a global defining the
    searchtype whenever we enter a new file or we enter new
    command and the file has changed, or we should look in the
    origin statement at this point. >  See <wgetfile> %
% search this file %
    IF ( movmkr _ lookup ($movmkr, $stmtnm, nametyp) ) # endfil
    THEN
       % we have found it %
       IF movmkr # wda.dacsp THEN
          % it is not the same as our current location %
          BEGIN
          cursptr _ movmkr; cursptr[1] _ 1;
          RETURN(TRUE);
          END
       ELSE IF NOT ( inhelp %OR athelp% ) THEN
```

```
                % check to see if the view is the same %
                IF NOT <samevw>(&vspstg, &wda) THEN
                   BEGIN
                   cursptr _ movmkr; cursptr[1] _ 1;
                   RETURN(TRUE);
                   END;
        % search file indexes %
           RETURN (<srchfndx>(&cursptr, $movmkr, $stmtnm, &wda));
        END.

% Sequence generator(s) for printg %
   (teedsw) PROCEDURE (sw, astrng);                                    9A
      % Given the address of a sequence work area, this routine changes
      its vspecs in accord with the specifications 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;
         REF sw, astrng;
         length _ astrng.L;
         count _ empty - 1;
         vs1 _ sw.swvspec;
         vs2 _ sw.swvsp2;
         UNTIL (count _ count+1) > length DO
            IF (char _ *astrng*[count]) = '; THEN
               BEGIN
               UNTIL (char _ *astrng*[count _ count + 1]) = '; DO
                  BEGIN
                  % skip over ca pattern. %
                  IF count >= length THEN EXIT LOOP1;
                  END;
               END
            ELSE vs1 _ setlt(char, vs1, vs2 : vs2);
         sw.swvspec _ vs1;
         sw.swvsp2 _ vs2;
         RETURN;
         END.


   (mencont) PROCEDURE (menumax, menusize, destsw, oldview);           9B
      % ? %
         REF menusw, destsw, oldview;
         IF moremenu THEN
            menumax _ menumax + menusize    % on with the next chunk %
         ELSE
            BEGIN
            IF &menusw THEN closeseq(&menusw := 0);
            destsw.swvspec _ oldview;
            destsw.swvsp2 _ oldview[1];
            destsw.swstid _ destsw.swcstid _ endfil;
            DROP(ALL);
            sport(&destsw);
            END;
         RETURN(menumax);
         END.
```

```
(ofstid) PROCEDURE (dpa, stnstg);                                    9C
   % evaluate  intra-file address expression %
   % Given the display area address in order to get the file number
   and a string containing an address expression, this routine will
   open the file, and return the corresponding stid and character
   count. %
      LOCAL vs1, vs2, cacode, useggen;
      LOCAL TEXT POINTER stptr, z1, z2;
      REF dpa, stnstg;
      stptr _ origin;
      stptr.stfile _ dpa.dacsp.stfile;
      FIND SF(*stnstg*) ^z1 SE(*stnstg*) ^z2;
      vs1 _ caddexp($z1, $z2, &dpa, $stptr : vs2, cacode, useggen);
      RETURN(stptr, stptr[1], vs1, vs2, cacode, useggen);
      END.

(qappender) PROCEDURE (string, appendstr, front, end, truncate,
dirflg, lines);                                                      9D
   % Appends contents of appendstr to front of string; puts contents
   delimited by front and end at the end of string.  (Does not try
   to append sppendstr if NULL; does not try to append front through
   end if front is zero.)  Does not reset original contents of the
   string. %
      LOCAL length;
      LOCAL TEXT POINTER tp;
      REF string, appendstr, front, end;
      IF appendstr.L THEN
         BEGIN
         *string* _ *appendstr*, *string*;
         appendstr.L _ 0;
         END;
      IF front THEN *string* _ *string*, front end;
      IF truncate  THEN
         BEGIN
         IF (length _ lines*72) < string.L THEN
            BEGIN
            string.L _ length;
            dirflg _ FALSE;
            END;
         IF FIND SF(*string*) [EOL] ^tp _tp THEN
            BEGIN
            *string*  _ SF(*string*) tp;
            dirflg _ FALSE;
            END;
         END;
      RETURN(dirflg);
      END.

(qcolumnate) PROCEDURE (sw, stid, columnatestr, selection,
flushflg);                                                           9F
   % query columnation %
   % Given the address of a sequence work area, the "current" line
   string, the selection string for the next item, and a flag for
   just flushing out the previous information, qcolumnate either
   does the simple flush, or adds the new item (sw.stid) to the
   columnated menu. %
```

```
    % Obeys global qcolwidth. %
    % NOTE:  sequence work area is destination -- PRIMARY one %
       REF sw, selection, columnatestr;
       LOCAL STRING names[30];
       LOCAL savlvl, qcolwork[7]; % temp READC work area %
       % see if time to pump out current line %
          IF columnatestr.L >= 72 OR flushflg THEN
             BEGIN
             savlvl _ sw.swclvl := sw.swslvl + 1;
             send(&sw, $columnatestr);
             sw.swclvl _ savlvl;
             *columnatestr* _ NULL;
             IF flushflg THEN RETURN;
             END;
       % construct new entry %
          % set up for and do name extraction %
             *names* _ NULL;
             qcolwork _ stid;
             qcolwork[1] _ 1; % point to first char %
             fechc1(forward, $qcolwork);
             xtrnam($names, $qcolwork, -1, 0);
          % add on to current line being built %
             *columnatestr* _ *columnatestr*, *selection*, *names*,
             *spacestr*[1 TO (qcolwidth - names.L - selection.L)];
       RETURN;
       END.

(qdirparse) PROCEDURE (start, end);                                    9F
    % parse query directives %
    % Given the address of a text pointer to the current position in
    the statement (which will NOT be changed by this procedure), the
    address of a text pointer which may be changed to point to the
    beginning of a qspec block or the end of the statement if no
    block exists, qdirparse locates the query directive string, if
    any, sets end, and returns TRUE and the count character of the
    next character to be scanned. %
    % If no directives are found, it returns FALSE and the count of
    the end of the statement. %
    % It attempts to do the scan quickly to optimize case where no
    directives exist. %
       LOCAL
          count,
          drswork[7];          % string work area for READC. %
       REF start, end;
       % Initialize the string work area. %
          drswork _ start;
          drswork[1] _ start[1];
          fechc1(forward, $drswork);
          count _ 0;
       % Find the first. %
          CASE READC($drswork) OF
             = "#:
                BEGIN
                IF NOT count THEN
                   BEGIN
                   IF READC($drswork) # "# THEN REPEAT CASE;
```

```
                    end[1] _ drswork[1] - 2;
                    BUMP count;
                    REPEAT CASE;
                    END
                 ELSE
                    BEGIN
                    IF READC($drswork) # "# THEN REPEAT CASE;
                    RETURN( TRUE, drswork[1]);
                    END;
                 END;
              = ENDCHR:
                 RETURN( FALSE, end[1] _ drswork[1]);
              ENDCASE REPEAT CASE;
        END.

(qinclude) PROCEDURE (sw, string, qlkleft, qlkright, topflag);      9G
    % process query link list (includes) %
    % Process from global qlkleft to qlkright, obeying user's
    viewspecs in links and the qspecs directives.  Each link in the
    list is executed, and a new sequence is opened at the new
    location. User viewspecs, if any, predominate over query
    defaults.
    The linked-to statement is included without its statement name
    and without any directives ("##....##") it may contain.  Qspecs
    govern the inclusion of the substructure as menu.
    Before leaving, the file and new sequence are closed.
       File not closed for now; need file management code to  take
       care of stids if files must be closed.
    PARAMETERS:  address of sequence work area (main), flag for
    including name on linked-to statement(s).  If topflag is true, we
    are processig the topnode or links in the top node.  This affects
    whether or not we will accept new qspecs.  Generally no if
    false-- formatting based on the top node. %
    % NOTE: assuming format of "##[..](..)(..)[..](..)etc##  If a
    queryspecs setting occurs at the end of the string, it will
    pertain to the substructure of this node (if applicable) since it
    stands alone.  This, of course, will be erased if another list
    occurs before substructure of this node processed.  To set qspecs
    to affect the sub of the node begin addressed, use a [..] last in
    the list (or only element). %
       LOCAL
          nofind,   % to avoid looping: TRUE if neither item located
          %
          stid;     % for new file/stmt %
       LOCAL TEXT POINTER
          curpos,    % current position in string being processed %
          oldpos,    % temp for position %
          startspec, % start of qspec or link %
          endspec;   % end of qspec or link %
       LOCAL STRING qstn[200]; % statement number %
       REF sw, string, qlkleft, qlkright;
       % ******* ON SIGNAL ELSE RETURN; ******* %
          % currently, this is a way out - lnkspec will complain when
          it finds no more links %
       INVOKE(didntwork, label6);
       % initialization %
```

```
      IF glkleft[1] = glkright[1] THEN
         BEGIN
      (label6):                                                    9G6G2
         DROP(ALL);
         RETURN;
         END;
      stid _ curpos _ oldpos _ glkleft;
         % in case no link, just viewspecs %
      curpos[1] _ endspec[1] _ glkleft[1];
      FIND glkright < 2CH $NP ^glkright;
         % strip garbage off end of link list %
   % run through list, getting [..] first, then (...)(...) etc. %
   LOOP
      BEGIN
      % process QSPECS   %
         nofind _ FALSE;
         IF FIND curpos > 2CH $NP ^endspec "[ ^startspec
         $(LD/"=)  % NOTE:  really ought to use FS and let
         anything go by, just trapping ] for this to handle
         errors in the best way % "] < CH ^endspec THEN
            BEGIN
               % found QSPECS %
            qsparse ($startspec, $endspec, topflag);
            curpos[1] _ endspec[1] + 1;
               % next char to process %
            END
         ELSE BUMP nofind;
         IF (oldpos[1] _ curpos[1]) >= glkright[1] THEN EXIT
         LOOP;
      % process next link %
         % Parse link and get stid. %
            glnkspec($curpos, $stno, $stn2 % filename % ,$qstn %
            number % , $num % vspecs % ); % this will go to err
            if no more links %
            IF stn2.L % filename exists % THEN
               stid _ fstid($str2) % open the file %
            ELSE stid _ ofstid( &qda, $qstn);
         % concatenate user string with ours (esb), ours first;
         change viewspecs only if not being menued.  Otherwise
         must only put out one line! %
            IF topflag THEN *num* _ "esb", *num*
            ELSE *num* _ "te"; % one line, one level in menu %
            feedsw(&sw, $num % vspecs % ); % put new viewspecs in
            the sequence work area %
         % remove directives, send out %
            qstrip(stid, &sw % destination of qstrip's output % ,
            topflag, &string);
         IF topflag AND qspecs.querinc THEN % will not be TRUE if
         NOT topflag (see qsparse) %
            BEGIN
            qmenu(stid, endfil, &sw); % do a menu if
            user-specified %
            END;
      % test for done %
         CASE curpos[1] OF
            = oldpos[1]: EXIT LOOP;
```

```
            % no more links in statement %
          >= qlkright[1]: EXIT LOOP;
            % end of section of stmt to process %
          ENDCASE nofind _ FALSE;
            % we found link %
        BUMP curpos[1];  % advance to next character %
      IF nofind THEN EXIT;
        % didn't find [...] or (...) - DB error.  Just ignore %
      END; % of qspecs/links processing loop %
    DROP(ALL);
    RETURN;
    END.

(qlnkspec) PROCEDURE (tptadr, usrstg, fnmstg, stnstg, vspstg);      9H
  % Makes use of the new lnkprs %
    LOCAL datastr[30];
    REF  usrstg, fnmstg, stnstg, vspstg, tptadr;
    lnkprs(&tptadr, $datastr);
    % Set up text pointers from datastr. %
      p1 _ datastr[us]; p1[1] _ datastr[us + 1];
      p2 _ datastr[ue]; p2[1] _ datastr[ue + 1];
      p3 _ datastr[fs]; p3[1] _ datastr[fs + 1];
      p4 _ datastr[fe]; p4[1] _ datastr[fe + 1];
      p5 _ datastr[ds]; p5[1] _ datastr[ds + 1];
      p6 _ datastr[de]; p6[1] _ datastr[de + 1];
      p7 _ datastr[vb]; p7[1] _ datastr[vb + 1];
      p8 _ datastr[ve]; p8[1] _ datastr[ve + 1];
      tptadr _ datastr[le]; tptadr[1] _ datastr[le + 1];
    *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 AND fnmstg.L # empty  THEN *stnstg* _ "0;
      % May really want it to be empty, e.g., to change viewspecs
      when not changing file %
    RETURN;
    END.

(qmenu) PROCEDURE (stid, endstd, destsw);                           9I
  % output a query menu %
  % Given the stid of the up of the menu and the end stid of the
  sequence, the main display area address, and a destination work
  area address, qmenu generates the output for the plex below (if
  any).
  Menu statements are defined as:
```

```
    All named statements
    All statements beginning with an asterisk
NOTE:  All other statements will appear exactly where they occur,
interspersed with menu selections, and at same level.
These are formatted so they become selectable by number.  Qmenu
attaches a number to each and pushes an address to that place (an
stid) onto the query state stack so that the mapping from option
selection to address in a file is easy.
QSpecs determine whether COLUMNATION is to be performed.
System globals:  qmenumax - max # simultaneous menu items
permitted.  qcolwidth = # horizontal increments to a menucolumn.
%
    LOCAL
        menusize,    % size of the menu before checking with user %
        menumax,     % local temp for max allowed items %
        code,        % result of testing for menu statement %
        stringadr,   % temp for calls %
        oldview [2],% temp for saving/restoring viewspecs %
        origlevel;   % level of the UP of this plex
            (for comparison to detect end) %
    LOCAL STRING
        columnatestr [75],  % one line %
        selection [20];
            % selectable number accompanying menu item %
    REF menusw, destsw;
    % initialization %
        menumax _ menusize _
            IF qspecs.queropts THEN qspecs.queropts ELSE qmenumax;
        *columnatestr* _ NULL;
    % open a new sequence for this menu plex - get first stid %
        &menusw _ openseq(stid, endstd,
            destsw.swvspec,
            destsw.swvsp2,
            0,          % no user sequence generator, please! %
            destsw.swcacode);
                % open a secondary sequence so we can control reading
                of meru plex right here %
        INVOKE(closeit);
        oldview _ destsw.swvspec;
        oldview[1] _ destsw.swvsp2;
        stid _ seqgen (&menusw);
            % throw away top statement - already processed %
        destsw.swvspec.vstrnc _ 1;
            % 1 line (truncated for simple menu case %
        origlevel _ menusw.swvspec.vslev _ menusw.swclvl + 1;
            % 1 level (truncated for simple menu case %
            % change to permit menued lower levels if desired. %
    % process the entire plex %
        % -----set viewspecs = 1 line %
        LOOP % once for each statement %
            BEGIN
            stid _ seqgen(&menusw); % next stid %
            % test for done (or no data) %
                IF stid = endfil THEN EXIT;
                CASE menusw.swclvl OF
                    < origlevel: EXIT LOOP; % done with menu plex %
```

```
            > origlevel: REPEAT LOOP; % sub of plex level %
            ENDCASE NULL; % fall through %
        % see if it's a menu statement %
          CASE code _ tstmenu(stid) OF
            = 1, = 2: % menu statements %
              BEGIN
              % test for too many menu items - exceeds system
              max %
                IF (qmenucnt _ qmenucnt + 1) > menumax AND
                nlmode=fulldisplay THEN
                  BEGIN
                    % Go back to interact with user; must
                    come back here to close menusw or
                    continue. %
                  destsw.swstid _ endfil;
                  DROP(ALL);
                  sport(&destsw);
                  menumax _
                    mencont(menumax, menusize, &destsw,
                    $oldview);
                  END;
              % Put the item on the menu stack. %
                IF newstk AND NOT pushent(stid, mentyp) THEN
                  BEGIN
                  % Cannot SIGNAL or use err because of
                  switched stacks.  Rather, set global
                  error flag to be TRUE, and send back
                  endfil. %
                  hseqer _ TRUE;
                  % send back EOF because we are done %
                    IF &menusw THEN closeseq(&menusw :=
                    0);
                    destsw.swcstid _ destsw.swstid _
                    endfil;
                    DROP(ALL);
                    sport(&destsw); % don't want to be
                    called again.  Finished with a query
                    branch %
                  END;
              % build selection number string %
                IF FIND SP(stid) SP THEN
                  *selection* _ STRING(qmenucnt), '.
                ELSE
                  *selection* _ STRING(qmenucnt), '., SP;
              IF qspecs.quercol THEN  % columnation requested
              %
                IF code = 2 THEN
                  BEGIN
                  % This is an unnamed, menued statement in
                  the middle of a columnated menu. %
                  % Flush out any current columnation stuff
                  and continue with the first line of this
                  menued, unnamed statement. %
                  IF columnatestr.L THEN
                    qcolumnate(&destsw, stid,
                    $columnatestr, $selection, TRUE);
```

```
                        % Continue processing. %
                        END
                    ELSE
                        BEGIN
                        gcolumnate(&destsw, stid, $columnatestr,
                        $selection, FALSE); % add this %
                        EXIT CASE;
                        END;
                % straight menu %
                    gstmt(&menusw, &destsw, FALSE, $selection);
                        % do the statement, executing INCLUDES,
                        no substructure, append "selection" to
                        front %
                IF overscreen THEN
                    BEGIN
                    menumax _
                        mencont(menumax, menusize, &destsw,
                        $oldview);
                    overscreen _ FALSE;    % reset overscreen %
                    END;
                END; % of menu-handling %
            = 3, = 5: % non-menu statements - may have
            directives %
                BEGIN
                    % flush out any current columnation stuff %
                    % process it, exclude substructure %
                stringadr _
                    IF code = 3 THEN $""
                    ELSE $selection;    % asterisk case %
                destsw.swvspec.vstrnc _ oldview.vstrnc;
                    % truncation may be on - make default number
                    of lines show for this case %
                gstmt(&menusw, &destsw, FALSE, stringadr);
                IF overscreen THEN
                    BEGIN
                    menumax _
                        mencont(menumax, menusize, &destsw,
                        $oldview);
                    overscreen _ FALSE;
                    END;
                destsw.swvspec.vstrnc _ 1;
                    % restore the temporary suppression of line
                    truncation disabled for non-named menu or
                    non-menu %
                END;
            = 0: NULL;          % watch out - error occurred %
            ENDCASE NULL;
        END;      % of menu plex loop %
    % cleanup %
    IF columnatestr.L THEN
        gcolumnate(&destsw, stid, $columnatestr, $selection,
        TRUE);
            % flush out possible incomplete line %
    IF &menusw THEN closeseq(&menusw := 0);
        % done with menu plex sequence.  Return to caller %
    destsw.swvspec _ oldview;
```

```
      destsw.swvsp2 _ oldview[1];
   DROP(ALL);
   RETURN;        % normal exit %
   (closeit) CATCHPHRASE ();                                    9I7J
      CASE SIGNALTYPE OF
         = aborttype:
            BEGIN
            IF &menusw THEN closeseg(&menusw := 0);
            CONTINUE;
            END;
         ENDCASE CONTINUE;
   END.

(queryseg) PROCEDURE (sw, phase);                               9J
   % query sequence generator program %
   % This controls the printout or display of a QUERY branch when a
   SHOW function is requested.  It handles INCLUDES, Q-specs, and
   menu generation. %
      LOCAL saveview;
      REF sw;
      CASE phase OF
         = sqopn: RETURN;     % OPEN %
         = sqgnxt: NULL;      % NEXT (below) %
         = sqcls: RETURN;     % CLOSE %
         ENDCASE err($"Query System Failure");
      % *******
         ON SIGNAL ELSE
            BEGIN
            % % Some awful failure; send back EOF with error flag
            set. % %
            hseger _ TRUE;   % % Error message will be wrong, but...
            % %
            sw.swcstid _ sw.swstid _ endfil;
            sport(&sw);      % % don't want to be called again.
            Finished with a query branch % %
            END; ******* %
      INVOKE(awfulerror, label4);
      qnewstmt.L _ 0;
      saveview _ sw.swvspec;
      qstmt(&sw,
         0,      % no previous sequence %
         TRUE,   % process the subplex of this statement, if any %
         $"");
            % process addressed node top statement and build a menu
            for any substructure %
      sw.swvspec _ saveview;
      sw.swcstid _ sw.swstid _ endfil;
         % send back EOF because we are done %
      (label4):                                                 9J2K
         DROP(ALL);
         sport(&sw);
            % don't want to be called again.  Finished with a query
            branch %
      (awfulerror) CATCHPHRASE ();                              9J2L
         CASE SIGNALTYPE OF
            = aborttype:
```

```
                BEGIN
                   % Some awful failure; send back EOF with error
                   flag set. %
                hseger _ TRUE;
                   % Error message will be wrong, but... %
                sw.swcstid _ sw.swstid _ endfil;
                TERMINATE;
                END;
             ENDCASE CONTINUE;
      END.

(qsender) PROCEDURE (string, workarea, topflag);                    9K
   % sends the contents of string out to sequence workarea with
   level controlled by topflag.  Resets contents of string after
   send. %
   % It avoids sending a null string %
      LOCAL savlvl;
      REF string, workarea;
      IF string.L THEN
         BEGIN
         savlvl _ workarea.swclvl :=
            workarea.swslvl + (IF topflag THEN 0 ELSE 1);
         send(&workarea, &string);
         workarea.swclvl _ savlvl;
         string.L _ 0;
         END;
      RETURN;
      END.

(qsparse) PROCEDURE (qspstrt, end, topflag);                        9L
   % set global queryspecs %
   % interpret string between qspstrt and end and set proper flags
   in global, qspecs - WARNING to data base builder - this is a
   hard-nosed son-of-a-bitch routine. %
      LOCAL
         char,    % next char to process %
         number,
         term, % number of characters to scan %
         qspwork[7]; % string work area %
      LOCAL STRING nmstr[10];
      REF qspstrt, end;
      % initialize string work area. %
         qspwork _ qspstrt;
         qspwork[1] _ qspstrt[1];
         fechc1( forward,  $qspwork);
         term _ end[1] - qspstrt[1];
      number _ FALSE;
      qspcreset(topflag); % set default values for qspecs %
      FOR term DOWN UNTIL <= 0 DO
         BEGIN
         CASE char _ READC($qspwork) OF    % next char of statement %
            ="c, ="C:
               IF topflag THEN qspecs.quercol _ TRUE;
            ="m, ="M:
               IF topflag THEN qspecs.querinc _ FALSE;
                  % Do not include substructure.-- Main node only %
```

```
            ='p, ='P:
               qspecs.querprt _ TRUE;
            ='n, ='N:
               number _ TRUE;
            ='=:
               IF NOT number:=number+1 THEN EXIT LOOP;
            = D:
               BEGIN          % digit %
               IF number # 2 THEN EXIT LOOP;
               IF nmstr.L = nmstr.M THEN EXIT LOOP; % avoid overflow
               %
               *nmstr* _ *nmstr*, char;
               END;
            = NP:
               REPEAT LOOP;
            ENDCASE EXIT LOOP;
         END; % LOOP %
      IF topflag AND nmstr.L THEN
         qspecs.queropts _ VALUE($nmstr);     % may truncate value! %
      RETURN;
      END.

(qspcreset) PROCEDURE (topflag);                                        9M
   % reset default queryspecs values %
   % qspecs is global %
      IF topflag THEN
         BEGIN
         qspecs.quercol _ FALSE;
         qspecs.queринc _ TRUE;
         qspecs.queropts _ FALSE;
         END;
      qspecs.querprt _ FALSE;
      RETURN;
      END.

(qstmt) PROCEDURE (sw, prevsw, topflag, appendstr);                     9M
   % process one QUERY source statement %
   % Examine "current" data base source statement, executing any
   links and performing INCLUDES wherever context and Q-specs
   dictate.  Trigger menu generation only if topflag=TRUE.  ELSE
   ignore substructure to this statement. %
   % It topflag = TRUE, assumed to be top addressed node, i.e. an
   asterisk as the first character must be stripped off %
   % qstmt will either be called (1) by queryseq to process the
   branch being SHOWN (addressed) (prevsw=0), or (2) by qmenu to
   process a single statement in the plex being menued (uncolumnated
   only) (prevsw#0) %
      LOCAL
         code,   % Type of statement: menu, non-menu, etc. %
         truncate,  % Do not execute links beyond the frist line if
         TRUE. %
         sendsw,   % temp for sequence work area address %
         nextchr,  % count of next character to be scanned. %
         dirflg;   % TRUE = q-directive found %
      LOCAL TEXT POINTER
         oldpos,   % front of this segment %
```

```
      endnew;   % end of this segment %
   REF sw, prevsw, sendsw, appendstr;
      % could be top node or a linked-to INCLUDED node %
   % general initialization %
      oldpos _ endnew _ sw.swcstid;
      endnew[1] _ oldpos[1] _ CASE code _ tstmenu(oldpos) OF
         = 3, = 4: 2; % Comment character to be peeled off %
         ENDCASE 1; % Logical front of this statement %
      truncate _
         IF NOT topflag AND code IN [1,2] THEN TRUE ELSE FALSE;
      &sendsw _ IF &prevsw THEN &prevsw ELSE &sw;
   LOOP
      BEGIN
         % Handle contents of this statement.  Optimize case
         where no directives occur.  No FIND statements used
         until queryspecs encountered.  If this is a top level
         node, process  until segments of text are sent and all
         link list INCLUDES have been taken.  If this is a menu
         item, gather data in global string for single send when
         includes have been taken; avoid CRs generated by sends.
         %
      dirflg _ qdirparse ($oldpos, $endnew : nextchr);
      dirflg _
         qappender($qnewstmt, &appendstr, $oldpos, $endnew,
         truncate, dirflg, sendsw.swvspec.vstrnc);
            % append and send previous segment if this is a top
            level node; otherwise, just append.  If this is a
            menu node, only append first 72 characters or one
            line; change dirflg to reflect limit reached to exit
            loop. %
      IF topflag OR NOT dirflg THEN       % send out now. %
         qsender($qnewstmt, &sendsw, topflag);
      IF NOT dirflg THEN EXIT LOOP;
      oldpos[1] _ endnew[1] := nextchr;
         % advance pointers for include %
      qinclude (&sendsw, $qnewstmt, $oldpos, $endnew, topflag);
         % do link list found by qdirparse - whether called with
         a menu statement or a main node.  qinclude calls qstrip
         to process included stmt, qstrip sends the stmt out
         without directives %
      oldpos[1] _ endnew[1] _ nextchr;
         % advance pointers for next scan %
      END;        % of text/include loop %
   IF topflag THEN qmenu (sw.swcstid, sw.swlbstid, &sendsw);
      % we are now processing the TOP statement (called from
      queryseq) and about to do MAIN subplex.  Do menu as
      requested %
   RETURN;
   END.   % normal exit %

(qstrip) PROCEDURE (stid, destsw, topflag, string);                    90
   % decrud a data base stmt %
   % THIS IS ALMOST IDENTICAL TO QSTMT!!! DROP ONE. %
   % Given the source sequence work area address, and the
   destination sequence work area address, drop off special query
   commands  and port-send the string out %
```

```
    LOCAL
       code,        % Type of statement: menu, non-menu, etc. %
       truncate,    % TRUE => do not execute links beyond first
       line %
       dirflg,      % TRUE if directive found %
       nextchr;     % curent next char to process %
    LOCAL TEXT POINTER
       oldpos,      % front of text to SEND %
       endnew;      % end of text to SEND %
    REF destsw, string;
    % initialize %
       oldpos _ endnew _  stid;
       oldpos[1] _ endnew[1] _
          CASE code _ tstmenu(stid) OF
             = 3, = 4: 2;  % Comment character to be peeled off %
             ENDCASE 1;    % Logical front of this statement %
       truncate _
          IF NOT topflag AND code IN [1,2] THEN TRUE ELSE FALSE;
    % Handle contents of this statement.  Optimize case where no
    directives occur.  No FIND statements used until queryspecs
    encountered. %
       LOOP
          BEGIN
             % keep processing through this stmt.  If this is a
             top level node, process  until segments of text are
             sent.  If this is a menu item, gather data in global
             string for single send when includes have been taken;
             avoid CRs generated by sends. %
          dirflg _ qdirparse ($oldpos, $endnew : nextchr);
          dirflg _
             qappender(&string, $"", $oldpos, $endnew, truncate,
             dirflg, destsw.swvspec.vstrnc);
             % append and send previous segment if this is top
             level node; otherwise, just append.  If this is a
             menu node, only append first 72 characters or one
             line; change dirflg to reflect limit reached to exit
             loop. %
          IF topflag THEN qsender(&string, &destsw, topflag);
             % send out now. %
          IF NOT dirflg THEN EXIT LOOP;
          oldpos[1] _ endnew[1] _ nextchr;
             % advance pointers for include checks and next scan %
          END;     % of text/include loop %
    RETURN;
    END.

(tstmenu) PROCEDURE (stid);                                    9P
    % test if query menu stmt %
    % RETURNs
    0: if ERROR encountered
    1: if statement represented by stid is named (therefore menu)
    2: if statement is an unnamed menued statement
    3: if statement represented by stid starts with non-menu symbol
    4: if statement begins with comment character
    5: No "non-menu" character defined.  This is non-menu, but no
    character peeled off. %
```

```
        LOCAL char, x;
        INVOKE(didntwork, return0);
           % WATCH OUT - GETNMF CALLS ERR with "Bad statement
           identifier" %
        IF NOT menchr OR NOT comchr THEN
           BEGIN
           x _ IF getnmf(stid) THEN 1 ELSE 5;
              % search for special comment or menu characters %
           DROP(ALL);
           RETURN(x);
           END;
        CCPOS SF(stid);
        CASE (char _ READC) OF
           =menchr: x _ 3;
           =comchr: x _ 4;
           ENDCASE
              x _ IF getnmf(stid) THEN 1 ELSE 2;
        DROP(ALL);
        RETURN(x);
        (return0):                                                    9P3H
           DROP(ALL);
           RETURN(0);
        END.

% Context stack building %
   (pushent) PROCEDURE (stid, type);                                  10A
      % Makes use of the global curstk as the context to be filled in %
      % Type = 1: mentyp; =2: multyp. %
        LOCAL stkad, itemad, wrkcnt, work;
        REF stkad, itemad;
        IF NOT curstk.stktyp THEN
           % get a new stack %
           BEGIN
           % get first block in this stack; link it to conrng %
           IF NOT (&stkad _ getblk(67B, &qstorblk)) THEN
              BEGIN %we are out of free storage - see if more is
              available%
              IF NOT (&qstorblk _ getblk(1777B , $dspblk)) THEN
              RETURN(FALSE);
              &qstorblk _ &qstorblk + bhl;
              makezone(&qstorblk, 70B, 70B, 1777B);
              IF NOT (&stkad _ getblk(67B, &qstorblk)) THEN
                 RETURN(FALSE);
              END;
           stkad[1] _ &qstorblk;
           curstk.constk _ &stkad _ &stkad + 2;
           stkad.conlnk _ -1;
           stkad.stkcnt _ 0;
           curstk.stktyp _ type;
           END
        ELSE
           BEGIN
           % Use totcnt to get to the proper block. %
           &stkad _ curstk.constk;
           FOR wrkcnt _ curstk.totcnt DOWN stkmax UNTIL <= stkmax DO
              BEGIN
```

```
            IF (&stkad _ stkad.conlnk)<=0 THEN RETURN(FALSE);
            END;
         END;
      % Put the entry on the stack. %
         IF (stkad.stkcnt_stkad.stkcnt+1) > stkmax THEN
            BEGIN
            % must get a new block %
            % Reset stkad.stkcnt %
               BUMP DOWN stkad.stkcnt;
            % Try to get a new block.  Put address in previous
            block's link field. %
            IF NOT (work _ getblk(67B, &qstorblk)) THEN
               BEGIN %we are out of free storage - see if more is
               available%
               IF NOT (&qstorblk _ getblk(1777B , $dspblk)) THEN
               RETURN(FALSE);
               &qstorblk _ &qstorblk + bhl;
               makezone(&qstorblk, 70B, 70B, 1777B);
               IF NOT (work _ getblk(67B, &qstorblk)) THEN
                  RETURN(FALSE);
               END;
            stkad.conlnk _ work + 2;
            &stkad _ work;
            stkad[1] _ &qstorblk;
            &stkad _ &stkad + 2;
            stkad.conlnk _ -1;
            stkad.stkcnt _ 1;
            END;
         &itemad _ &stkad + stkad.stkcnt*2;
         itemad _ stid;
         itemad[1] _ IF getnmf(stid) THEN getnam(stid)
            ELSE 0; % zero if no name %
         BUMP curstk.totcnt;
      RETURN(TRUE);
      END.

FINISH
```

(adstr)	<nine, psprograms, 01130>	LOCAL	4D14A1
(ccmpile)	<nine, psprograms, 0373>	PROCEDURE	4B2
(ccompile)	<nine, psprograms, 0212>	PROCEDURE	4B1
(cinsprog)	<nine, psprograms, 01459>	PROCEDURE	7B
(cpcaseq)	<nine, psprograms, 0796>	PROCEDURE	4D1
(cpchkstate)	<nine, psprograms, 0819>	PROCEDURE	4D2
(cpcleanup)	<nine, psprograms, 0865>	PROCEDURE	4D3
(cperror)	<nine, psprograms, 0875>	PROCEDURE	4D4
(cpgtjfn)	<nine, psprograms, 0502>	PROCEDURE	4C3
(cphaltf)	<nine, psprograms, 0538>	PROCEDURE	4C4
(cpjfns)	<nine, psprograms, 0557>	PROCEDURE	4C5
(cpload)	<nine, psprograms, 0903>	PROCEDURE	4D5
(cpmkseq)	<nine, psprograms, 0937>	PROCEDURE	4D6
(cpopenf)	<nine, psprograms, 0582>	PROCEDURE	4C6
(cppmap)	<nine, psprograms, 0601>	PROCEDURE	4C7
(cpprcseq)	<nine, psprograms, 0959>	PROCEDURE	4D7
(cpprfix)	<nine, psprograms, 0686>	PROCEDURE	4C9
(cppsout)	<nine, psprograms, 0707>	PROCEDURE	4C10
(cpputxt)	<nine, psprograms, 0999>	PROCEDURE	4D8
(cprcrd)	<nine, psprograms, 0473>	PROCEDURE	4B3
(cpreset)	<nine, psprograms, 0734>	PROCEDURE	4C11
(cprjfn)	<nine, psprograms, 0751>	PROCEDURE	4C12
(cpsizef)	<nine, psprograms, 0775>	PROCEDURE	4C13
(dinstupg)	<nine, psprograms, 01374>	PROCEDURE	5B3
(gpbsz)	<nine, psprograms, 01301>	PROCEDURE	5B1
(gpexpgm)	<nine, psprograms, 01659>	PROCEDURE	10B1
(gpgmrst)	<nine, psprograms, 01390>	PROCEDURE	5B4
(gppop)	<nine, psprograms, 01319>	PROCEDURE	5B2
(gpstatus)	<nine, psprograms, 01757>	PROCEDURE	12B1
(jsystype)	<nine, psprograms, 01022>	PROCEDURE	4D9
(marksym)	<nine, psprograms, 01118>	PROCEDURE	4D13
(movwrds)	<nine, psprograms, 0673>	PROCEDURE	4C8
(popsym)	<nine, psprograms, 01114>	PROCEDURE	4D12
(programs)	<nine, psprograms, 047>	EXT	2E
(prscpostr)	<nine, psprograms, 01042>	PROCEDURE	4D10
(rcrdlink)	<nine, psprograms, 01069>	PROCEDURE	4D11
(upgcnv)	<nine, psprograms, 01574>	PROCEDURE	8B1
(upgjfn)	<nine, psprograms, 01128>	LOCAL	4D14
(upgsstr)	<nine, psprograms, 01781>	PROCEDURE	12B1N
(xcompile)	<nine, psprograms, 0120>	PROCEDURE	4A3B
(xcompile)	<nine, psprograms, 067>	PROCEDURE	4A1A
(xcprcrd)	<nine, psprograms, 0110>	PROCEDURE	4A1B
(xpdelnstitute)	<nine, psprograms, 01410>	LOCAL	6A1
(xpdelete)	<nine, psprograms, 01257>	PROCEDURE	5A1
(xpinsert)	<nine, psprograms, 01437>	LOCAL	7A
(xpinstitute)	<nine, psprograms, 01543>	LOCAL	8A1
(xpreset)	<nine, psprograms, 01612>	LOCAL	9A1
(xprun)	<nine, psprograms, 01646>	LOCAL	10A1
(xpset)	<nine, psprograms, 01672>	LOCAL	11A
(xpshow)	<nine, psprograms, 01731>	LOCAL	12A1

```
< NINE, PSPROGRAMS.NLS;31, >, 2-May-78 17:45 BLP ;;;;
FILE psprograms % ( arcsubsys,x110,) (arcsubsys,1109,)
(reinine,PSPROGRAMS.rel,) %
   NOMESS
   ALLOW! %system calls: for encapsulating %
%declarations%
   REF cpsw;
   DECLARE CONSTANT % %
      % command word %
         statement = 29 ,                                          2B1A
         file = 51 ,
         program = 52,
         ca = 53,
         procedure = 54,
         grammar = 55,
      % compile states %
         cpstart = 0,                                              2B2A
         cpinpt = 1,                                               2B2B
         cpoutpt = 2,                                              2B2C
         cpname = 3,                                               2B2D
         cpfile = 4,                                               2B2E
         cpWorking = 5,                                            2B2F
         cperrmsg = 6,                                             2B2G
         cpfin = 7,                                                2B2H
         cpnmwds = 8,                                              2B2I
      % global support %
         cpilev = 3,            % interrupt level for compilation % 2B3A
         cpchan = 34,           % channel for compiler jsys traps % 2B3B
         cpijfn = 777B ;              % special jfn for file to
         compile %                                                 2B3C
   DECLARE %special string %
      rdelim = (5000001B, 21B10),      % right delimiter = " %
      ldelim = (5000002B, 104B7);      % <NULL> " %
   DECLARE % jsys handling correspondence table %
      cptrap = (
         % dont handle the following
            aic, chfdb, clzff, debrk, eir, idtim, nout, runtm, sevec,
            sir, sout, time
         %
         $gtjfn, $cpgtjfn,
         $haltf, $cphaltf,
         $jfns,  $cpjfns,
         $openf, $cpopenf,
         $pmap,  $cppmap,
         $psout, $cppsout,
         147B, $cpreset,  % JSYS RESET %
         $rljfn, $cprljfn,
         $sizef, $cpsizef,
         -1, -1);
   (programs) EXTERNAL _ (                                         2E
      $"XLOAD", $xload,
      $"XCOMPILE", $xcompile,
      $"XCPRCRD", $xcprcrd,
      $"XPINSERT", $xpinsert,
      $"XPSHOW", $xpshow,
      $"XPSET", $xpset,
```

```
      $"XPRESET", $xpreset,
      $"XPDELETE", $xpdelete,
      $"XPRUN", $xprun,
      $"XPINSTITUTE", $xpinstitute,
      $"XPDEINSTITUTE", $xpdeinstitute,
      $"XHANDLE", $xhandle,
      0,0);
%compile-time switches%
   SET NSW = FALSE;  %TRUE => compile to  (relnine,wmpsprograms.rel,)%
%compile%                                                              4
   %compile PCP interface routine%
      %-NSW%                                                           4A1
         (xcompile)     % compile xroutine %
         PROCEDURE(type, locptr, compptr, fnamptr, diagnos, vs,
         windid);                                                      4A1A
            % Procedure description
               FUNCTION
                  Set up the necessary parameters and call the core
                  routine
               ARGUMENTS
                  type -- what to compile
                  locptr -- where to find it
                  compptr -- what compiler to use
                  fnamptr -- where to put the result
                  diagnos -- where to put diagnostics
               LOCALS
                  da    -- display area for last window
                  strtstid -- stid to start compilation from
                  msgdest -- where to put messages (see ccmpile)
                  errs -- number of compilation errors
               RESULTS
                  none
               NON-STANDARD CONTROL
                  none
               GLOBALS
                  none
               %
            REF type, locptr, compptr, fnamptr, diagnos, vs;
            LOCAL da REF, strtstid, msgdest, errs, wordcount;
            LOCAL STRING work[59];
            % prepare the arguments %
               &type _ ELEM #type#[ccwtype];
               strtstid _ [ELEM #locptr#[tppair]];
               IF &compptr THEN &compptr _ ELEM #compptr#[tppair];
               IF &fnamptr THEN &fnamptr _ ELEM #fnamptr#[tppair];
               &da _  [findwa( windid )].widdarea;
               msgdest _ IF &diagnos THEN [ELEM #diagnos#[tppair]] ELSE
               0;
               IF msgdest > 0 THEN dpset(dspstrc, msgdest, endfil,
               endfil);
            % call the compile core routine %
               errs _ ccompile(&type, strtstid, &compptr, &fnamptr,
               msgdest, &vs, &da : wordcount );
            % check if <^O> during compile %
               IF inptrf THEN rstcntlo(); % tell FE to resume normal
               output %
```

```
      % handle final messages %
         IF errs THEN
            *work* _ EOL, "Unsuccessfully Finished (",
            STRING(errs), " Errors)", EOL
         ELSE % no errors %
            *work* _ EOL, "Successfully Finished (",
            STRING(wordcount, 8), "B Words)", EOL;
         dismes(1, $work );
         IF errs THEN typelit(1, $""); % ask for confirmation %
      RETURN;
      END.

   (xcprcrd)                % record compilation xroutine %
   PROCEDURE( diagnos REF );                                    4A1B
      LOCAL stid;
      stid _ [ELEM #diagnos#[tppair]];
      IF NOT cprcrd(stid) THEN
         dismes( 1, $"Unable to get recording" );
      RETURN;
      END.

%-NSW%                                                          4A2
%+NSW%                                                          4A3
   % this is the old one %
   (xcompile) PROCEDURE   % compile program %                   4A3B
      %FORMALS% (
         entity REF,       %entity command word%
         locptr REF,       %pointer to selected location to begin%
         compptr REF,      %pointer to name of compiler%
         fnameptr REF      %pointer to name of output file%
         );
      LOCAL
         retry, jfn,
         location REF, compiler REF, fname REF,
         adstr[40],        % data structure for link parsing %
         savecsp, % save location for current csp %
         errors,  % error count %
         tptr2 REF,        % local ptr to text ptr %
         da REF;           % ptr to display area %
      LOCAL STRING
         loccompiler[200],% local compiler string %
         complstr[200],   % compiler string %
         errstr[200],     % error string %
         locoutf[200],    % local output rel-file name %
         rfilname[200];   % NSW output rel-file name %
      LOCAL TEXT POINTER
         tp1, tp2;
      %------------------------%
      IF &locptr THEN &location _ ELEM #locptr#[tppair];
      IF &compptr THEN &compiler _ ELEM #compptr#[tppair];
      &fname _ ELEM #fnameptr#[tppair];
      &da _ lda(ELEM #locptr#[wndw]);
      CASE ELEM #entity#[cwtype] OF
         = 5i %- file -%: % compile file %
            BEGIN
            % put file names into local strings %
```

```
            lnbfls( &compiler, 0, $complstr);
            CASE lnbfls( &frame, 0, $rfilname) OF
               = lhostn: NULL;
               ENDCASE
                  ABORT(noremote, $"Remote File Manipulations
                  Not Implemented Yet");
         % generate local output file name %
            FIND SF(*rfilname*) ^tp1 SE(*rfilname*) ^tp2;
            FIND ". ^tp2;
            FIND [".] > CH ^tp1;
            *locout* _ tp1 tp2, ".rel";
         % retrieve compiler from NSW %
            % extension must be "sav" %
            FIND  SF(*complstr*) ^tp1 SE(*complstr*) ^tp2;
            FIND ". ^tp2;
            IF NOT FIND $SP ("vas") THEN *complstr* _ tp1 tp2,
            ".sav";
            IF NOT wmopen(0, $complstr,0, 0, $loccompiler, 0)
            THEN err($"Cannot find compiler in NSW");
         % set dacsp in the display area record and invoke the
         compiler %
            savecsp _ da.dacsp;
            da.dacsp _ location;
            errors _ cpcmpfl( TRUE, $loccompiler, $locout,
            &da);
            da.dacsp _ savecsp;
         % tell the user about any errors %
            IF errors > 0 THEN
               BEGIN
               *errstr* _ STRING(errors), " error(s):  Type
               CA.";
               dismes (2, $errstr);
               END;
         % deliver output file to NSW %
            IF NOT wmdeliver(0, $locout, $rfilname, 0) THEN
            err($"cannot deliver compiler output file");
         % delete local compiler, output files %
            R1 _ jfn _ sgtjfn(0, $loccompiler, $errstr);
            IF NOT SKIP !JSYS delf THEN *errstr* _ *errstr*,
            EOL, "Cannot delete local compiler";
            R1 _ jfn _ sgtjfn(0, $locout, $errstr);
            IF NOT SKIP !JSYS delf THEN *errstr* _ *errstr*,
            EOL, "Cannot delete local output file";
            IF err.L THEN err($errstr);
         END;
      = 52 %- l10 -%:  % L10 User Program %
         BEGIN
         gpl10( location, &da );
         END;
      = 53 %- content -%:  % analyzer filter %
         BEGIN
         da.dacacode _ cpconan(&location, &da);
         da.davspec.vscapf _ TRUE;
         END;
      = 54 %- procedure -%:  % Compile procedure %
         BEGIN
```

```
            cpcmproc( location, &da );
            END;
         ENDCASE
         IF NOT donthelp THEN
            BEGIN
            retry _ HELP (badarg, $"The type argument was
            incorrect.  The allowed values are File, L10, or
            Content.", cindex);
            REPEAT CASE (retry);
            END
         ELSE ABORT (badarg, $"The type argument was
         incorrect.");
      RETURN;
      END.

   %+NSW%                                                          4A4
%compile core routines%
   (ccompile)        % compile higher level core routine %
   PROCEDURE(type, strtstid, compptr REF, fnamptr REF, msgdest, vs
   REF, da REF);                                                   4B1
      % Procedure description
         FUNCTION
            Set up the necessary parameters and call the lower level
            core routine for actual compilation.  This procedure
            distinguishes between the entities being compiled:
               File:
                  No special action taken.
               Program:
                  Load the file after compilation
               Grammar:
                  Use CML compiler and compact grammar after
                  compilation
               Content analyzer:
                  Load and institute the pattern as content analyzer
                  after compilation.
               Procedure:
                  Replace the procedure after compilation.
            All temporary files and old REL versions will be deleted
            when ccompile terminates.
         ARGUMENTS
            type -- what to compile
            strtstid -- where to start compilation
            compptr -- what compiler to use
            fnamptr -- where to put the result
            msgdest -- where to put diagnostics
            vs -- viewspecs to use or 0
            da -- display area pointer
         LOCALS
            cacod    -- address of content analyzer
            usqcod   -- address of sequence generator
            outjfn   -- jfn for compiler output file
            rcrdflg -- record/no record flag (see ccmpile)
            errors   -- number of compiler erors
            extn     -- extension for output file
            proc     -- procedure to call when compilation is done
            procparam2 -- 2nd parameter to proc (1st is always
```

```
        adstr)
        delflg    -- delete output file flag
        adstr     -- data structure for link arsing
        cmptp     -- text pointer to compiler name in link syntax
        linktp    -- text pointer to linkstr
        linkstr -- output filename in link synatx
        filstr    -- output file name
     RESULTS
        Number of compilation errors.
     NON-STANDARD CONTROL
        signals created when output files cannot be opened or
        program names not found
     GLOBALS
        none
     %
  LOCAL cacod, usqcod, outjfn, rcrdflg, errors, extn REF, proc
  REF, delflg, procparam2, wordcount, adstr[40];
  LOCAL TEXT POINTER tp1, tp2, linktp, cmptp;
  LOCAL STRING filstr[99], linkstr[39], locstr[1999];
  % prepare default arguments %
     FIND SF(*linkstr*) ^linktp SF(*110link*) ^cmptp;
     IF NOT &compptr THEN &compptr _ $cmptp;
     IF NOT &fnamptr THEN &fnamptr _ $linktp;
     IF &vs THEN (cacod, usqcod) _ (da.dacacode, da.dausqcod)
     ELSE (cacod, usqcod) _ (0, 0);
     rcrdflg _ (msgdest <= 0);
     &extn _ $relext;
  % prepare arguments for the various cases %
     CASE type OF
        = file: NULL;
        = program:
           BEGIN
              IF NOT FIND
                 SF(strtstid) ["PROGRAM"/"FILE"] $NP ^tp1 1$LD
                 ^tp2
              THEN ABORT(upgerr, $"Couldn't find program name");
              *linkstr* _ "<, tp1 tp2, ",>";
           END;
        = grammar:
           BEGIN
              FIND SF(*cmllink*) ^cmptp;
              &compptr _ $cmptp;
              &extn _ $cgrext;
           END;
        = ca:
           BEGIN
              FIND strtstid > $NP ^tp1 $5PT ^tp2;
              BUMP upgcacnt;
              *linkstr* _ "<, "up", STRING(upgcacnt), %tp1 tp2,%
              ",>";
              &extn _ $caext;
              usqcod _ $cpcaseq;
              msgdest _ -1; %no messages%
              rcrdflg _ FALSE;
           END;
        = procedure:
```

```
            BEGIN
               IF NOT FIND
                  SF(strtstid)  [*(J $NP ^tp1 1$LD ^tp2 $NP *)
                  [*PROC*]
               THEN ABORT(upgerr, $"Couldn't find procedure
               name");
               *linkstr* _ *<, tp1 tp2, ",>";
               &extn _ $procext;
               usgcod _ $cpprcseg;
               msgdest _ 0; %feedback window%
               rcrdflg _ FALSE;
            END;
         ENDCASE;
      % protect against errors %
         INVOKE(cpxcatch);
      % get output file jfn %
         CASE lnbfls(&fnamptr, 0, $filstr) OF
            = lhostn: NULL;
         ENDCASE
            ABORT(noremote, $"Remote file manipulation not yet
            impelemented");
         IF NOT (outjfn _ lgetjfn( 0, $filstr, &extn, gtjoosf, $lit
         )) THEN
            ABORT( ofilerr, $lit );
      % go to do the job %
         errors _ ccmpile( strtstid, &compptr, outjfn, &vs, cacod,
         usgcod, msgdest, rcrdflg : wordcount );
      % close and don't release output jfn %
         sysclose(outjfn+4B11, $lit);
      % drop the signals %
         DROP(cpxcatch);
      % prepare default parameters for final operations %
         lnkprs( &fnamptr, $adstr );
         &proc _ FALSE;
         delflg _ FALSE; %default on file retention%
         procparam2 _ TRUE;
      % final operations: loading, etc. if necessary %
         IF errors THEN delflg _ TRUE
         ELSE CASE type OF
            = file: NULL;
            = ca:
               BEGIN
                  &proc _ $cpload;
                  delflg _ TRUE;
                  procparam2 _ FALSE;
               END;
            = program,
            = procedure:
               BEGIN
                  &proc _ $cpload;
                  delflg _ TRUE;
               END;
            = grammar: NULL;
         ENDCASE;
         % call final procedure %
            *locstr* _ NULL;
```

```
        IF &proc THEN errors _ errors + proc( $adstr,
        procparam2, $locstr );
        IF locstr.L THEN cpputxt($cpmsgdest, 0, $locstr);
    % get rid of output jfn %
      IF delflg THEN sysdelete( outjfn ) %delete the file%
      ELSE % delete old version %
        BEGIN
          delovsrns(outjfn, 1);
          reljfn(outjfn);
        END;
    RETURN( errors, wordcount );
    % catchphrase definition %
      (cpxcatch) CATCHPHRASE();                                    4B1P1
        BEGIN
        CASE SIGNALTYPE OF
          = aborttype :
            BEGIN
              DISABLE(cpxcatch);
              IF outjfn THEN sysdelete(outjfn:=0);
            END;
          ENDCASE;
        CONTINUE;
        END;
  END.

(ccmpile)          % compile lower level core routine %
PROCEDURE(strtstid, cmpptr, outjfn, vs, cacode, usqcod, msgdest,
rcrdflg);                                                          4B2
  % Procedure description
    FUNCTION
      Set up the necessary parameters and start up the
      COMPILER fork
    ARGUMENTS
      strtstid -- starting stid for compilation
      cmpptr -- text pointer to the link specifying the
      compiler
      outjfn -- jfn for output file (not opened)
      vs --
        = 0: include all levels
        # 0: viewspecs to use
      cacode -- address of content analyzer
      usqcode -- address of sequence generator
      msgdest -- diagnostics output destination
        = 0: feedback window
        = -1: no diagnostics output
        > 0: stid for recording in nls file
      rcrdflg -- sequential recording flag
        = TRUE: recording requested
        = FALSE: no recording necessary
    RESULTS
      number of compilation errors
    NON-STANDARD CONTROL
      err called
    GLOBALS
      cpfkh, fmtstid, cpmsgdest, cpstate, cppgcount, cplstflno
    %
```

```
REF cmpptr, diagnos, vs;
LOCAL compjfn, adstr[40];
LOCAL STRING filstr[100], locstr[100] ;
% protect against errors %
   INVOKE(cpcatch);
% get the compiler jfn %
   lnkprs( &cmpptr, $adstr );
   IF NOT (compjfn _ upgjfn( $adstr )) THEN
      err($"Compiler not found");
   jfntostr( compjfn, $locstr, 100B6 );
   IF *locstr* # *savext* AND *locstr* # *exeext* THEN
      err($"Illegal compiler name");
% create the source sequence %
   &cpsw _ cpmkseq( strtstid, &vs, cacode, usqcod );
% initialize parameters and globals %
   IF (fmtstid _ seqgen( &cpsw )) = endfil THEN err($"empty
   file");
   fmtstid[1] _ 1;
   cplstflno _ 0;
   cppgcount _ 1;
   cpstate _ cpstart;
   cpmsgdest _ msgdest;
   cpojfn _ outjfn;
   *leftover* _ NULL;
% handle sequential recording file %
   IF rcrdflg THEN % sequential recording needed %
      BEGIN
         *filstr* _ *cprcdname*, *initsr*;
         IF cprcdjfn _ lgetjfn($userstr, $filstr, $txtext,
         1810 %tmp%, $lit) THEN % open and clean it %
            BEGIN
               chfdbjsys(12B, cprcdjfn, -1, 0);  %pretend it's
               empty%
               IF NOT sysopen( cprcdjfn, readwrite, chrtyp,
               $lit) THEN
                  reljfn( cprcdjfn := 0 );
            END;
      END
   ELSE %sequential recording not needed %
      cprcdjfn _ 0;
% create fork and enable all capabilities %
   cpfkh _ crfork();
% get the save file %
   gtfkfile( cpfkh, compjfn );
% insert header %
   *filstr* _ "Compilation of ";
   IF fmtstid.stfile THEN filnam( fmtstid.stfile, $filstr );
   *filstr* _ *filstr*, " By ", *initsr*, " On ";
   getdat( $filstr );
   cpputxt( $cpmsgdest, -1, $filstr );
% define and set interrupts and traps %
   definetraps( $cptrap );
   settraps( cpfkh, cpchan, cpilev );
% startup the fork %
   IF cpmsgdest < 0 THEN dismes(1, $"
   compiling");
```

```
      strtfk( cpfkh, 0 );
   % in the meantime wait until it finishes %
      waitfork(cpfkh);
   % release compiler jfn %
      reljfn( compjfn );
   % cleanup %
      cpcleanup();
   % drop the catchphrase %
      DROP(cpcatch);
   RETURN( cpercount, cpwdcount );
   % catchphrase definition %
      (cpcatch) CATCHPHRASE();                                    4B2T1
         BEGIN
         CASE SIGNALTYPE OF
            = aborttype :
               BEGIN
                  DISABLE(cpcatch);
                  cpcleanup();
               END;
            ENDCASE;
         CONTINUE;
         END;
   END.

(cprcrd)          % record the sequential recording file %
PROCEDURE( stid );                                                4B3
   LOCAL tmpjfn;
   LOCAL STRING filstr[100];
   % prepare the recordin filename %
      *filstr* _ *cprcdname*, *initsr*;
   IF tmpjfn _ lgetjfn($userstr, $filstr, $txtext, 1B11 %old%,
   $lit) THEN
      BEGIN % insert sequential %
         dpset(dspstrc, stid, endfil, endfil);
         jfntostr( tmpjfn, $filstr, 0 );
         reljfn( tmpjfn );
         inseqn( stid, -1, $filstr, FALSE );
      END
   ELSE % no sequential file %
      RETURN(FALSE);
   RETURN(TRUE);
   END.

% encapsulating -- jsys traps %
   % General comment for JSYS handlers
      JSYS handlers are passed four arguments:
         A fork handle
         The current JSYS number
         The address where the encapsultor saves the current fork PC
         The adress of a 15-word block containing the content of the
         fork's registers
      JSYS handlers return two boolean values:
         handled:
            TRUE if JSYS has been taken care of
            FALSE let TENEX handle it
         unfreeze:
```

```
            TRUE to unfreeze the fork FALSE to leave it frozen
    %
    (cpgtjfn)           % gtjfn handler %
    PROCEDURE( fkhndl, jsysnum, frkpc, infreg );                    4C3
       % Procedure description
          FUNCTION
             Provide a jfn to inferior fork according to the
             following:
                cpijfn: when the file to compile is requested
                cpojfn: when the output file is requested
          ARGUMENTS
             See general JSYS handling comment
          RESULTS
             See general JSYS handling comment
          NON-STANDARD CONTROL
             cperror is called
          GLOBALS
             none
          %
       REF frkpc, infreg;
       LOCAL jfnword, block[10] ;
       LOCAL STRING work[50];
       IF (infreg[1] .A 1000000B) THEN %short gtjfn%
          jfnword _ infreg[2]
       ELSE % long gtjfn %
          BEGIN
             gtfblk( $block, fkhndl, infreg[1], 10 );
             jfnword _ block[1];
          END;
       IF jfnWord.LH # 100B THEN RETURN(FALSE, TRUE);
       cpchkstate();
       jsystype($"gtjfn"); % DEBUG %
       infreg[1] _
          CASE cpstate OF
             = cpinpt: cpijfn;
             =cpoutpt: cpojfn;
          ENDCASE cperror($"Unknown Compiler File Request");
       skipreturn( &frkpc );
       RETURN( TRUE, TRUE );
       END.

    (cphaltf) % haltf handler %
    PROCEDURE( fkhndl, jsysnum, frkpc, infreg );                    4C4
       % Procedure description
          FUNCTION
             type a terminating message to the user
          ARGUMENTS
             See general JSYS handling comment
          RESULTS
             See general JSYS handling comment
          NON-STANDARD CONTROL
             none
          GLOBALS
             none
          %
       REF frkpc, infreg;
```

```
    LOCAL mestyp;
    jsystype($"haltf"); % DEBUG %
    cpercount _ infreg[1]; % remeber number of errors %
    RETURN( FALSE, TRUE );
    END.

(cpjfns)  % jfns handler %
PROCEDURE( fkhndl, jsysnum, frkpc, infreg );                    4C5
    % Procedure description
        FUNCTION
            Let 10X handle the JSYS except for the source file
            (cpijfn) in which case supply the inferior fork with
            dummy strings as file name defaults
        ARGUMENTS
            See general JSYS handling comment
        RESULTS
            See general JSYS handling comment
        NON-STANDARD CONTROL
            none
        GLOBALS
            none
        %
    REF frkpc, infreg;
    LOCAL STRING work[20];
    jsystype($"jfns"); % DEBUG %
    IF infreg[2].RH # cpijfn THEN RETURN(FALSE, TRUE);
    CASE infreg[3].LH OF
        = 1000B: *work* _ "A";
        = 0: *work* _ "A.REL";
    ENDCASE *work* _ NULL;
    *work* _ *work*, 0;
    ptfstr( $work, fkhndl, infreg[1] );
    RETURN( TRUE, TRUE );
    END.

(cpopenf) % openf handler %
PROCEDURE( fkhndl, jsysnum, frkpc, infreg );                    4C6
    % Procedure description
        FUNCTION
            Let 10X handle the JSYS except for the source file
            (cpijfn) in which case nothing is done (skip return)
        ARGUMENTS
            See general JSYS handling comment
        RESULTS
            See general JSYS handling comment
        NON-STANDARD CONTROL
            none
        GLOBALS
            none
        %
    REF frkpc, infreg;
    jsystype($"openf"); % DEBUG %
    IF infreg[1].RH # cpijfn THEN RETURN(FALSE, TRUE);
    skipreturn( &frkpc );
    RETURN( TRUE, TRUE );
    END.
```

```
(cppmap)  % pmap handler %
PROCEDURE ( fkhndl, jsysnum, frkpc, infreg );                    4C7
   % Procedure description
      FUNCTION
         Map a source page to compiler's address space. Do
         nothing for non source or map-out
      ARGUMENTS
         See general JSYS handling comment
      RESULTS
         See general JSYS handling comment
      NON-STANDARD CONTROL
         none
      GLOBALS
         fmtstid, cplstflno, leftover
      %
   LOCAL myadrs, maxwords, numwords, i;
   REF myadrs, frkpc, infreg ;
   LOCAL STRING stdtxt[20], locstr[2000];
   LOCAL locbuf[512];
   % return if not for input file %
      IF infreg[1].LH # cpijfn THEN RETURN( FALSE, TRUE );
   % initialize %
      jsystype($"pmap");  % DEBUG %
      &myadrs _ $locbuf;
      maxwords _ 512;
      *locstr* _ *leftover*;
      *leftover* _ NULL;
   % format as many statements as possible %
      WHILE fmtstid # endfil DO %format the statement%
         BEGIN
            IF fmtstid[1] = 1 THEN % add prefix code for stid %
               cpprfix( fmtstid, $stdtxt )
            ELSE % clear the prefix %
               *stdtxt* _ NULL;
            % make a statement in locstr %
               *locstr* _ *locstr*, *stdtxt*, fmtstid
               SE(fmtstid), EOL;
            % move full words from locastr to myadrs %
               numwords _ movwrds( $locstr, &myadrs, maxwords );
               &myadrs _ &myadrs + numwords;
               maxwords _ maxwords - numwords;
            % check if page full %
               IF maxwords <= 0 THEN EXIT LOOP;
            % get the next statement in line %
               fmtstid _ seqgen( &cpsw );  fmtstid[1] _ 1;
         END;
   % clean up rest of text and parameters %
      IF maxwords > 0 THEN % file ended %
         BEGIN
            % add a few NULLs to avoid garbage %
               *locstr* _ *locstr*, 0, 0, 0, 0;
               locstr.L _ locstr.L - 4;
            % transfer the last word (if any) %
               myadrs _ locstr[1];
               BUMP &myadrs;
```

```
                BUMP DOWN maxwords;
              % pad the page with zeroes %
                FOR i _ 0 UP UNTIL  >= maxwords DO myadrs[i] _ 0;
           END
        ELSE % page full %
           BEGIN
              % update fmtstid and leftover %
                FIND SE(fmtstid) ^fmtstid;
                IF locstr.L
                AND (fmtstid[1] _ fmtstid[1] + 1 %EOL% - locstr.L)
                < 2 THEN
                   BEGIN % something left over %
                      *leftover* _ *locstr*[1 TO 2-fmtstid[1]];
                      fmtstid[1] _ 2;
                   END;
           END;
     % put into fork's space %
        ptfblk( $locbuf, fkhndl, infreg[2].RH*10008, 512 );
     % and return %
        RETURN( TRUE, TRUE );
     END.

(movwrds) % move words from string to block %
PROCEDURE( astr REF, addres REF, uplimit );                          4C8
     LOCAL numwords;
     % evaluate the number of words to transfer %
        numwords _ MIN(uplimit, astr.L/5);
     IF numwords THEN % transfer words %
        BEGIN
           % move full words to addres %
              mvbfbf( &astr+1, &addres, numwords );
           % update the string %
              *astr*[1 TO numwords*5] _ NULL;
        END;
     RETURN( numwords );
     END.

(cpprfix) % create prefix for statement %
PROCEDURE( stid, str REF );                                          4C9
     LOCAL newjfn, fl REF;
     % no prefix for a-strings %
        IF stid.stastr THEN
           BEGIN
              *str* _ NULL;
              RETURN;
           END;
     % initialize prefix %
        *str* _ *ldelim*, STRING(stid.RH, 8), ":, STRING(stid.LH,
        8);
     % check if this is from a new file %
        IF stid.stfile # cplstflno THEN % new file %
           BEGIN
              &fl _ flntadr(cplstflno _ stid.stfile);
              newjfn _ IF filepart[cplstflno] THEN fl.flpart ELSE
              fl.florig;
              *str* _ *str*, "=, STRING(newjfn,8), *rdelim*;
```

```
            END
         ELSE % old file %
            *str* _ *str*, *rdelim*;
      RETURN;
      END.

   (cppsout) % psout handler %
   PROCEDURE( fkhndl, jsysnum, frkpc, infreg );                    4C10
      % Procedure description
         FUNCTION
            move the string to "cpostr" and check for state change
         ARGUMENTS
            See general JSYS handling comment
         RESULTS
            See general JSYS handling comment
         NON-STANDARD CONTROL
            none
         GLOBALS
            cpostr, cmpoflag
         %
      REF frkpc, infreg;
      LOCAL TEXT POINTER tp1, tp2;
      % copy the string %
         gtfstr( $cpostr, fkhndl, infreg[1] );
      % trim invisibles %
         IF FIND SE(*cpostr*) $NP ^tp2 PT SF(tp2) $NP ^tp1 THEN
         %assign line%
            *cpostr* _ tp1 tp2
         ELSE %line empty%
            *cpostr* _ NULL;
      % check and update state %
         jsystype($"psout"); % DEBUG %
         IF cpostr.L THEN cpchkstate();
      RETURN( TRUE, TRUE );
      END.

   (cpreset) % reset handler %
   PROCEDURE( fkhndl, jsysnum, frkpc, infreg );                    4C11
      % Procedure description
         FUNCTION
            simulate jsys as NOP
         ARGUMENTS
            See general JSYS handling comment
         RESULTS
            See general JSYS handling comment
         NON-STANDARD CONTROL
            none
         GLOBALS
            none
         %
      REF frkpc, infreg;
      jsystype($"reset"); % DEBUG %
      RETURN( TRUE, TRUE );
      END.

   (cprljfn) % rljfn handler %
```

```
   PROCEDURE( fkhndl, jsysnum, frkpc, infreg );                    4C12
      % Procedure description
         FUNCTION
            simulate jsys as NOP
         ARGUMENTS
            See general JSYS handling comment
         RESULTS
            See general JSYS handling comment
         NON-STANDARD CONTROL
            none
         GLOBALS
            none
         %
      REF frkpc, infreg;
      jsystype($"rljfn"); % DEBUG %
      CASE infreg[1] OF
         = cpijfn, = -1: %all not open%
            BEGIN
               skipreturn(&frkpc);
               RETURN( TRUE, TRUE );
            END;
      ENDCASE %other rljfn%
         RETURN( FALSE, TRUE );
      END.

   (cpsizef) % sizef handler %
   PROCEDURE( fkhndl, jsysnum, frkpc, infreg );                    4C13
      % Procedure description
         FUNCTION
            Let 10X handle the JSYS except for the source file
            (cpijfn) in which casea dummy page count is returned
            (skip return)
         ARGUMENTS
            See general JSYS handling comment
         RESULTS
            See general JSYS handling comment
         NON-STANDARD CONTROL
            none
         GLOBALS
            none
         %
      REF frkpc, infreg;
      jsystype($"sizef"); % DEBUG %
      IF infreg[1].RH # cpijfn THEN RETURN(FALSE, TRUE);
      infreg[2] _ (IF fmtstid = endfil AND NOT leftover.L THEN
      cppgcount ELSE (cppgcount _ cppgcount+1))*2560;
      skipreturn( &frkpc );
      RETURN( TRUE, TRUE );
      END.

%compile support routines%
   (cpcaseq) % Compile Content Analyzer Seq Generator %
   PROCEDURE (sw REF, which);                                      4D1
      LOCAL STRING locstr[200];
      CASE which OF
         =sqgnxt: % called for next in seq %
```

```
            BEGIN
               % send FILE and PROCEDURE statments %
                  *locstr* _ "up", STRING(upgcacnt);
                  *locstr* _ "FILE ", *locstr*, "
                  (", *locstr*, ") PROCEDURE;
                  IF FIND ";
                  send( &sw, $locstr );
               % send the actual pattern %
                  sw.swstid _ sw.swcstid;
                  sport( &sw );
               % send the RETURN and FINISH statements %
                  *locstr* _
                     " THEN RETURN(TRUE) ELSE RETURN(FALSE);
                     END.
                     FINISH ";
                  send( &sw, $locstr );
               % send an ENDFIL %
                  send( &sw, endfil );
            END;
         ENDCASE;
      RETURN;
      END.

   (cpchkstate)      % check and possibly move states %
   PROCEDURE;                                                    4D2
      % Procedure description
         FUNCTION
            Check the most recent output from the compiler and
            possibly perform a state transition.  In some cases
            other procedure to analyze the output and update globals
            are called
         ARGUMENTS
            none
         RESULTS
            none
         NON-STANDARD CONTROL
            cperror is called
         GLOBALS
            cpstate
         %
      LOCAL tmpstate;
      LOCAL TEXT POINTER tp1;
      tmpstate _ prscpostr();
      CASE cpstate OF
         = cpstart:
            IF tmpstate = cpinpt THEN cpstate _ cpinpt;
         = cpinpt:
            IF tmpstate = cpoutpt THEN cpstate _ cpoutpt;
         = cpoutpt:
            BEGIN
               IF tmpstate = cpinpt THEN %compiler couldn't open
               file%
                  cperror($"Couldn't open output file");
               cpputxt( $cpmsgdest, 0, $cpostr );
               IF tmpstate = cpfile THEN cpstate _ cpworking;
            END;
```

```
        = cpworking:
           BEGIN
              CASE tmpstate OF
                 = cperrmsg: rcrdlink();
                 = cpfin: cpstate _ cpfin;
              ENDCASE
                 cpputxt( $cpmsgdest, 0, $cpostr );
           END;
        = cpfin:
           BEGIN
              cpputxt( $cpmsgdest, 0, $cpostr );
              IF tmpstate = cpnmwds THEN % record the number of
              words %
                 BEGIN
                    cpstate _ cpnmwds;
                    FIND SF(*cpostr*) $D ^tp1;
                    *lit* _ SF(*cpostr*) tp1;
                    cpwdcount _ VALUE($lit, 8);
                 END;
           END;
        = cpnmwds: NULL;
     ENDCASE cperror($"Unidentified compiler output");
     *cpostr* _ NULL;
     RETURN;
     END.

  (cpcleanup)        % clean up after compiler's operation %
  PROCEDURE;                                                    4D3
     LOCAL STRING locstr[100];
     % close recording file %
        IF cprcdjfn THEN sysclose( cprcdjfn := 0, $locstr );
     % close the sequence %
        IF &cpsw THEN closeseq( &cpsw := 0 );
     % reset interrupt trapping %
        rstjtinterrupt( cpchan );
     RETURN;
     END.

  (cperror)          % compile error handler %
  PROCEDURE( errstr REF );  % No return from here... %          4D4
     % Procedure description
        FUNCTION
           This procedure aborts the compiler. It kills the fork,
           types a message, and calls err.
        ARGUMENTS
           none
        RESULTS
           none
        NON-STANDARD CONTROL
           err is called
        GLOBALS
           none
        %
     LOCAL STRING work[100];
     % kill fork %
        killfork( cpfkh := 0 );
```

```
    % change debreak address and go there %
      chgdbk( cpilev, $errdbk );
      !debrk();
      (errdbk):                                                    4D4D3
    % clean up trapping %
      cpcleanup();
    % create error message %
      *work* _ EOL, "Compiler error in state ", STRING(cpstate),
      EOL, *errstr*;
    % generate signal %
      err( $work);
    RETURN;
    END.

(cpload)  % load file after compilation %
PROCEDURE( adstr REF, msgflg, errstr REF );                        4D5
    % Procedure description
      FUNCTION
        this procedure interfaces to the loader by catching the
        appropriate signals
      ARGUMENTS
        none
      RESULTS
        number of errors (0 - successful, 1 - failure)
      NON-STANDARD CONTROL
        none
      GLOBALS
        none
      %
    LOCAL errors _ 0, strptr REF _ 0;                              4D5B
    % invoke the catchphrase %
      INVOKE(cpldctch, anycase);                                   4D5C1
    % call the loader %
      ldprog( &adstr, msgflg, &errstr );                           4D5D1
    % come back here in any case %
      (anycase): DROP(cpldctch);
      IF &strptr THEN *errstr* _ *errstr*, *strptr*;               4D5E2
    RETURN( errors );                                              4D5F
    (cpldctch) CATCHPHRASE(: &strptr);                             4D5G
      BEGIN
      CASE SIGNALTYPE OF
        = aborttype :
          BEGIN
            errors _ 1;
            TERMINATE;
          END;
        ENDCASE;
      CONTINUE;
      END;
    END.
                                                                   4D5H
(cpmkseq)          % make a sequence for the compiler %
PROCEDURE( strtstid, vs, cacode, usgcod );                         4D6
    LOCAL vsptr, vspec[2], stid2, sw;
    REF vs, vsptr, sw;
    % prepare the viewspecs %
```

```
      IF &vs THEN % user specified vs %
         &vsptr _ &vs
      ELSE % include all levels (possibly filtered) %
         BEGIN
            vspec _ vspec[1] _ 0;
            &vsptr _ $vspec;
            vsptr.vslev _ vsptr.vstrnc _ vsptr.vsnamf _
            vsptr.vsdaft _ vsptr.vsindf _ -1;
            vsptr.vscapf _ (cacode # 0);
            vsptr.vsusqf _ (usqcod # 0);
         END;
   % compute begin and end stids %
      stid2 _
         IF strtstid.stastr THEN strtstid
         ELSE seqend(strtstid, vsptr, vsptr[1]);
   % create the sequence %
      &sw _ openseq(strtstid, stid2, vsptr, vsptr[1], usqcod,
      cacode );
   RETURN( &sw );
   END.

(cpprcseq)        % Compile Procedure Seq Generator %
PROCEDURE (sw REF, which);                                    4D7
   LOCAL stid, vspec[2];
   LOCAL TEXT POINTER tp1, tp2;
   LOCAL STRING locstr[200];
   REF cpauxsw;
   CASE which OF
      =sqopn: % called at open %
         BEGIN
            % make auxiliary sequence %
               vspec _ vspec[1] _ 0;
               vspec.vslev _ vspec.vstrnc _ vspec.vsnamf _
               vspec.vsdaft _ vspec.vsindf _ -1;
               vspec.vsbrof _ TRUE; % PROCEDURE must be a branch
               %
               &cpauxsw _ openseq( sw.swcstid, sw.swcstid, vspec,
               vspec[1], 0, 0 );
         END;
      =sqgnxt: % called for next in seq %
         BEGIN
            % first time around %
               % send FILE and ALLOW statements %
                  stid _ sw.swstid;
                  IF NOT FIND SF(stid) $NP "( ^tp1 1$LD ^tp2
                  ["PROC"] THEN err(upgerr, $"Procedure name not
                  found");
                  *locstr* _ "FILE ", tp1 tp2, " ALLOW! ";
                  send( &sw, $locstr );
               % restore the sequence work area pointer %
                  sw.swstid _ sw.swcstid;
            % all the rest %
               LOOP BEGIN
                  IF (sw.swstid _ seqgen(&cpauxsw) ) = endfil
                  THEN
                     send( &sw, $"FINISH");
```

```
                    cpysw( &cpauxsw, &sw);
                    sport( &sw );
                 END;
            END;
         =sgcls: % called at close %
            BEGIN
               closeseg( &cpauxsw := 0 );
            END;
         ENDCASE err($"Illegal call to a seq gen program"); % called
         for any other purpose -- error %
      RETURN;
      END.

   (cpputxt)          % give user the text %
   PROCEDURE( msgdest, level, str );                                  4D8
      REF msgdest, str;
      IF NOT str.L THEN RETURN;
      CASE msgdest OF
         = 0: % show in feedback window %
            BEGIN
               *lit* _ EOL, *str*;
               typelit( 0, $lit );
            END;
         = -1: % don't display %
            NULL;
      ENDCASE % as a statement %
         BEGIN
            msgdest _ cisstr( msgdest, &str, level );
            dpset(dspstrc, msgdest, endfil, endfil );
         END;
      IF cprcdjfn THEN % record in sequential file %
         BEGIN
            *lit* _ *str*, CR, LF, CR, LF, 0;
            !sout( cprcdjfn, chbmty+$lit, 0);
         END;
      RETURN( msgdest );
      END.

   (jsystype)         % type trap diagnostics %
   PROCEDURE( string );                                               4D9
      % Procedure description
         FUNCTION
            type diagnostics -- for debug purposes only
         ARGUMENTS
            string -- JSYS name
         RESULTS
            none
         NON-STANDARD CONTROL
            none
         GLOBALS
            none
         %
      LOCAL STRING work[150];
      REF string;
      RETURN;   % for debug replace this by !JFCL 0 %
      *work* _ EOL, *string*, " trapped in state ", STRING(cpstate),
```

```
    EOL ,"cpostr: ", *cpostr*;
    dismes(1, $work);
    RETURN;
    END.

(prscpostr)        % parse *cpostr* %
PROCEDURE;                                                    4D10
    % Procedure description
        FUNCTION
            This procedure parses the "cpostr" from the compiler and
            "suggests" the new "cpstate". It returns zero if no
            guess can be made.
        ARGUMENTS
            none
        RESULTS
            A "suggested" new state.
        NON-STANDARD CONTROL
            none
        GLOBALS
            none
        %
    LOCAL newstate;
    LOCAL TEXT POINTER tp;
    IF cpostr.L = 0 THEN RETURN(cpstate);
    FIND SF(*cpostr*) ^tp;
    newstate _
        CASE TRUE OF
            = (FIND tp "FINISHED"): cpfin;
            = (FIND tp "-INPUT"): cpinpt;
            = (FIND tp "-OUTPUT"): cpoutpt;
            = (FIND tp "-FILE"): cpfile;
            = (FIND tp 1$D "B WORDS"): cpnmwds;
        ENDCASE cperrmsg;
    RETURN(newstate);
    END.

(rcrdlink)         % record an erroneous line and link %
PROCEDURE;                                                    4D11
    % Procedure description
        FUNCTION
            Record the link at the end of cpmsgdest and the
            erroneous line following it
        ARGUMENTS
            none
        RESULTS
            none
        NON-STANDARD CONTROL
            none
        GLOBALS
            cpmsgdest
        %
    LOCAL stid;
    LOCAL TEXT POINTER tp1, tp2, tp3, tp4, tp5, tp6;
    LOCAL STRING work[300];
    % parse the text line %
        IF NOT FIND
```

```
            SE(*cpostr*) $NP "" (1$D "=/TRUE) ^tp4 1$D ^tp3 ": ^tp2
            1$D ^tp1 "" $NP ^tp6 SF(*cpostr*) ^tp5
         THEN % no link found %
            BEGIN
               cpputxt( $cpmsgdest, 0, $cpostr );
               RETURN;
            END;
      % compute the stid %
         *work* _ tp1 tp2;
         stid.RH _ VALUE( $work, 8 );
         *work* _ tp3 tp4;
         stid.LH _ VALUE( $work, 8 );
      % create a link to the error %
         % initialize output string %
            *work* _ NULL;
         % append the filename %
            filnam(stid.stfile, $work);
         % insert the infile adress %
            FIND SF(*work*) ["(>"] ^tp1;
            ST tp1 tp1 _ "0, STRING( getsid(stid) );
      % give user the erroneous line %
         *work* _ tp5 tp6, "  ", *work*;
         % get read of "LINE" spec %
            IF FIND SF(*work*) ["LINE"] ^tp2 < 4CH ^tp1 tp2 > $NP
            1$D $NP ^tp2 THEN
               ST tp1 tp2 _ NULL;
         cpputxt( $cpmsgdest, 0, $work );
      RETURN;
      END.

   (popsym)   PROCEDURE;  % pops block of symbols from DDT's symbol
   table %                                                        4D12
      ddtpop();
      RETURN;
      END.

   (marksym)  PROCEDURE   % marks DDT's symbol table %            4D13
      (blocknm);   % 0 or string containing the block name %
                                                                  4D13A
      LOCAL STRING str[50];
      REF blocknm;
      IF &blocknm THEN *str* _ *blocknm*
      ELSE *str* _ "LOCAL";
      astruc( $str ); % force to upper case %
      ddtmark($str);
      RETURN;
      END.

   (upgjfn)  % get a jfn for parsed link data structure %        4D14
      PROCEDURE
         (adstr);    % address of parsed link data structure % 4D14A1
      LOCAL try, jfn;
      LOCAL TEXT POINTER tp1, tp2;
      LOCAL STRING filename[200], dirname[100];
      REF adstr;
```

```
% ALGORITHM %
   % if a directory name is specified for the link then this
   directory is searched for the file.  if no directory name
   is specified then the following directories are searched
   for the file (in the following order):
      NETSYS
      SUBSYS
      the default directory for links for the link
      the connected directory
      the login directory
   in all cases the default extension name is "SAV". %
% RETURNS %
   % this routine will return the jfn for the found file or
   FALSE;  it will generate an err for links which specify a
   remote hostname %

% get file name as specified in link (& check for local host)
%
   CASE lnbfls( 0, $adstr, $filename) OF
      = lhostn: NULL;
      ENDCASE err($"Remote File Manipulations Not Implemented
      Yet");
% only try once if directory specified in link %
   IF adstr[ue+1] > adstr[us+1] THEN
      BEGIN
      IF NOT (jfn _ lgetjfn(0, $filename, $savext, gtjprf,
      $lit) ) THEN
         IF NOT (jfn _ lgetjfn(0, $filename, $exeext, gtjprf,
         $lit) ) THEN
            err($lit);
      RETURN(jfn);
      END;
% now try various directories %
   tp1 _ adstr[fs];  tp1[1] _ adstr[fs+1];
   tp2 _ adstr[fe];  tp2[1] _ adstr[fe+1];
   *filename* _ tp1 tp2;
   try _ 0;
   *dirname* _ "NETSYS";
   LOOP
      CASE (jfn_
      lgetjfn($dirname,$filename,$savext,gtjprf,$lit)) OF
         = FALSE:
            CASE (jfn_
            lgetjfn($dirname,$filename,$exeext,gtjprf,$lit))
            OF
               = FALSE:
                  CASE try _ try + 1 OF
                     = 1:      % SUBSYS or NETSYS or whatever
                     %
                      *dirname* _ *subdir*;
                     = 2:      % SUBSYS %
                      *dirname* _ *subdr2*;
                     = 3:      % default directory for links %
                      IF NOT adstr[lfn] OR NOT gdftdir(
                      adstr[lfn], $dirname) THEN
                      ELSE REPEAT CASE;
```

```
                                = 4:      % connected directory %
                                   *dirname* _ NULL;
                                = 5:      % login directory %
                                   *dirname* _ *userstr*;
                                ENDCASE err($lit);
                            ENDCASE RETURN( jfn );
                        ENDCASE RETURN( jfn );
        END.

%delete%
   %delete PCP interface routine%
      (xpdelete) %Execute Delete Command%
      PROCEDURE (                                                    5A1
         %FORMALS%
            entity REF,          %entity type%
            rtnlist REF % result list %
            );
         LOCAL STRING filstr[200], badmsg[250];
         LOCAL retry;
         %----------------------%
         #rtnlist#[1] _ NULL;  % initialize result to no grammar name %
         CASE ELEM #entity#[CWtype] OF
            = 52 %- allprograms -%:     %SHOULD CALL WORKS MANAGER%
               BEGIN
               gpgmrst ();
               IF upgbsz > $upgbdf THEN gpbsz($upgbdf);
               upgcacnt _ 0; % reset CA count %
               END;
            = 56 %- lastprogram -%:     %SHOULD CALL WORKS MANAGER%
               BEGIN
               % must setup string to pass to FE to delete the grammar
               %
               gppop($filstr); % return grammar name, if any %
               IF filstr.L THEN #rtnlist#[1] _ *filstr*;
               END;
            ENDCASE
            IF NOT donthelp THEN
               BEGIN
               *badmsg* _ "The type argument was incorrect.  The
               allowed values are Allprograms, Last ";
               retry _ HELP (badarg, $badmsg, cindex);
               REPEAT CASE (retry);
               END
            ELSE ABORT (badarg, $"The type argument was incorrect.");
         RETURN(TRUE);
         END.

   %delete support routines%
      (gpbsz)  PROCEDURE  % Goto User program Buffer Size Set %      5B1
         (size);  % Number of pages to be in new buffer %          5B1A

         % RETURNS %
            % returns false for invalid request or address of first new
            page allocated %
         LOCAL stadr;
```

```
    IF rfpmax - size < minfpages OR (512*size) <
    (upgffbuf-upgbuf+1)
       THEN RETURN (FALSE);
    closeall();
    upgbsz _ size;
    rfpmin _ size + 1;  % Index to first file core page %
    stadr _ upgbend + 1;
    upgbend _ upgbuf + 512*size - 1;
    openall();
    RETURN (stadr);
    END.

(gppop)  % does Goto Program Pop a program %
PROCEDURE(progname REF);                                        5B2
    % Delete top program on user program stack --
       pop DDT's symbol table
       find any references to it and delete them
       give back its space to the buffer
       fix the stack pointer and the string of program names
       pass name of subsystem back for grammar deletion by FE %
    LOCAL oldend, i, value, bpa;
    LOCAL TEXT POINTER tp, tp1,tp2;
    IF &progname THEN *progname* _ NULL;  % pass string only if a
    subsys %
    IF upgskix <= 0 THEN ABORT (upgerr, S"stack is empty");
    % move thru program string %
       FIND SE(*upgnms*) $NP ^tp2 $PT ^tp1 $NP ^tp;
    % detach and delete the subsystem %
       INVOKE (sigdrop, gpgt1);
       IF (value _ upgstk[upgskix].LH := 0) THEN
          BEGIN % this was a subsys, pass back grammar name %
          IF &progname THEN getpkgname(value,&progname);
          delpkg(value);
          END;
       (gpgt1): DROP (sigdrop);                                 5B2G3
    popsym ();
    dinstupg (upgstk[upgskix]);
    IF upgstk[upgskix] IN [upgbuf, upgbend) THEN
       % if program is in buffer, free the buffer space %
       oldend _ upgffbuf := upgstk[upgskix];
    BUMP DOWN upgskix;
    % clear any breakpoints here %
       FOR i _ 0 UP UNTIL >= 10 DO
          IF bpa _ findbp(i) THEN
             IF ([[bpa]].LH = 265B3 % JSP %) AND
                ([bpa].RH IN [upgffbuf, oldend]) THEN
                   BEGIN
                   [[bpa]] _ [bpa+1];
                   [bpa] _ [bpa+1] _ 0;
                   END;
    % do procedure backups for any procedures in this program %
       FOR i _ 0 UP 2 UNTIL >= 10 DO
          IF proctbl[i] THEN
             IF ([proctbl[i]].LH = 254B3) AND
                ([proctbl[i]].RH IN [upgffbuf, oldend]) THEN
                   BEGIN
```

```
                    [proctbl[i]] _ proctbl[i+1];
                    proctbl[i] _ 0;
                    END;
        *upgnms* _ SF(*upgnms*) tp;
        RETURN;

        (sigdrop) CATCHPHRASE;                                          5B20
           TERMINATE;

        END.

     (dinstupg) PROCEDURE   % Deinstitute a program %                    5B3
        (pgmaddr);   % address of of the user program %                  5B3A
        % "deinstitute" the specified user program --
           find any references to it and delete them, but don't give
           back its space to the buffer -- thus it can "reinstituted"
           %
        LOCAL da;
        REF da;
        FOR &da _ $dpyare UP dal UNTIL >= $dpyare + dacnt*dal DO
           IF da.daexis THEN
              BEGIN  % fields pointing to program to be popped are
              reset %
              IF da.dacacode = pgmaddr THEN da.dacacode _ 0;
              IF da.dausqcod = pgmaddr THEN da.dausqcod _ 0;
              IF da.daukeycod = pgmaddr THEN da.daukeycod _ 0;
              END;
        RETURN;
        END.

     (gpgmrst)  PROCEDURE;  % Goto User program reset %                  584
        % reset the stack and all display area descriptor fields
        having to do with user programs %
        %reset user programs buffer size to default value%
        INVOKE (upgrtn, RETURN);
        LOOP gppop(FALSE);
        DROP(upgrtn);
        RETURN;

        (upgrtn) CATCHPHRASE;                                           5B4H
           BEGIN
           DISABLE (upgrtn);
           CASE SIGNAL OF
              = upgerr: TERMINATE;
              ENDCASE CONTINUE;
           END;

        END.

%deinstitute%
   %deinstitute PCP interface routine%
      (xpdeinstitute)   % Disestablish a user program %                  6A1
        PROCEDURE (type REF, window);
        LOCAL retry, t2ptr REF, da REF, field;
        LOCAL STRING nmstr[50];
        % decode type to determine field %
```

```
        CASE ELEM #type#[cwtype] OF
           = 53 %- content -%:       % content analyzer program %
              field _ dacacode;
           = 51 %- seggenerator -%: % sequence generator program %
              field _ dausqcod;
           = 52 %- sort -%:          % sort key program %
              field _ daukeycod;
        ENDCASE
        IF NOT donthelp THEN
           BEGIN
           retry _ HELP(badarg, $"The type argument was incorrect.
           The allowed values are Content, Seggenerator, or Sort",
           cindex);
           REPEAT CASE (retry);
           END
        ELSE ABORT (badarg, $"The type argument was incorrect.");
     % deinstitute program from current display area %
        &da _ lda(window);
        da.field _ 0;
     % make sure we dont undo this in cmdfinish %
        cspupdate _ FALSE;
     RETURN ;
     END.

% insert program structure %
  (xpinsert)    %Execute Insert Command%                                  7A
     PROCEDURE (
        %FORMALS%
           entity REF,          %entity type%
           destptr REF,         %destination pointer%
           level,               %level adjustment characters%
           sourceptr REF,       %source pointer%
           subentity REF        %subentity type%
           );
     LOCAL
        destination REF, source REF;
     %----------------------%
     IF &sourceptr THEN &source _ ELEM #sourceptr#[tppair];
     IF &destptr THEN &destination _ ELEM #destptr#[tppair];
     CASE ELEM #entity#[cwtype] OF
        = 60: % programmer's entity %
           % source holds prog. entity type block %
           cinsprog(&subentity, &destination, &source, level);
        ENDCASE
           ABORT (badarg, $"The type argument was incorrect.");
     RETURN;
     END.

  (cinsprog)   % CL: ;  insert a program structure %
  PROCEDURE (type REF LIST, destination REF, source REF, level);      7B
     % Procedure description
        FUNCTION
           Insert a programmer's entity, which may be text (comments)
           or a plex copied from the current programmer's template
           file.  This file has branches with names equal to the
           entity.  The file is stored in the useroptions file and can
```

```
        be altered thru the useroptions subsys.
    ARGUMENTS
        type--REF-pointer to block holding command word, CW string.
        destination-REF-pointer to destination text ptr,
            DSEL(#"CHAR" or "STATEMENT")
        source--REF- pointer to text ptr pair for LSEL(#"TEXT"),
        used for comments
        level--INT-relative level count.
    RESULTS
        proc-value
    NON-STANDARD CONTROL
        none
    GLOBALS
        none
    %
  % Declarations %
    LOCAL STRING
        tmpstr[80];
    LOCAL
        dtemp1 REF, dtemp2 REF,
        stemp1 REF, stemp2 REF,
        i,
        typetpp REF,
        stpblk[4];  % for src txt ptr pair %
    LOCAL TEXT POINTER
        srctp1, srctp2,
        tps, tpe;
  % decode formals %
    &typetpp _ ELEM #type#[tppair];
    CASE ELEM #type#[cwtype] OF
        = 103:  % insert comment %
            BEGIN
            % setup text pointers %
                &stemp1 _ &source;
                &stemp2 _ &stemp1 + d2sel;
                &dtemp1 _ &destination;
                &dtemp2 _ &dtemp1 + d2sel;
            % insert the text, adding comment delimiters %
                IF NOT nortnrings THEN
                    clist(ctcmk, destination.stfile, nofile);
                IF NOT nodisplay THEN
                    dpset(dsprfmt, destination, endfil, destination);
                ST dtemp2 _
                    SF(dtemp2) dtemp2, " % ", stemp1 stemp2, " %",
                    dtemp2 SE(dtemp2);
                curmkr _ dtemp2;
                curmkr[1] _ dtemp2[1] + stemp2[1]-stemp1[1] + 4;
                IF NOT nortnrings THEN clupdt();
            END;
        ENDCASE   % program entities to copy from template file %
            BEGIN
            % get txt ptrs to entity string %
                &stemp1 _ &typetpp;
                &stemp2 _ &stemp1 + d2sel;
            % get src stid for template file %
                *tmpstr* _ "(, *tmpfile*[2 TO (tmpfile.L - 2)], ", ,
```

```
                stemp1 stemp2, ".d)";
                FIND SF(*tmpstr*) ^tps SE(*tmpstr*) ^tpe;
                srctp1 _ curmkr;
                srctp1[1] _ curmkr[1];
                caddexp($tps, $tpe, dsparea(cwndow), $srctp1);
                % now srctp1 holds new text pointer %
                stpblk _ plxset(srctp1 : stpblk[d2sel]);  % get stids
                of head, tail %
                stpblk[1] _ stpblk[d2sel + 1] _ 1;
             % copy the branch %
                curmkr _ xcmgrp(&destination, level, $stpblk,
                copyflag, 0);
                curmkr[1] _ 1;
             END;
      % Return %
         RETURN;
      END.

%institute%
   %institute PCP interface routine%
      (xpinstitute)     % Establish a user program %                    8A1
         PROCEDURE (type REF, prognameptr REF, window);
         LOCAL retry, progname REF, t2ptr REF, field, da REF;
         LOCAL STRING nmstr[50];
         % move program name into local string %
            &progname _ ELEM #prognameptr#[tppair];
            &t2ptr _ &progname + d2sel;
            *nmstr* _ progname t2ptr;
         % decode type to determine field %
            CASE ELEM #type#[cwtype] OF
               = 53 %- content -%:       % content analyzer program %
                  field _ dacacode;
               = 51 %- seggenerator -%: % sequence generator program %
                  field _ dausgcod;
               = 52 %- sort -%:          % sort key program %
                  field _ daukeycod;
            ENDCASE
               IF NOT donthelp THEN
                  BEGIN
                  retry _ HELP(badarg, $"The type argument was
                  incorrect.  The allowed values are Content,
                  Seggenerator, or Sort.", cindex);
                  REPEAT CASE (retry);
                  END
               ELSE ABORT (badarg, $"The type argument was
               incorrect.");
         % institute program into current display area %
            &da _ lda(window);
            da.field _ upgcnv( $nmstr );
         % make sure we dont undo this in cmdfinish %
            cspupdate _ FALSE;
         RETURN;
         END.

   %establish support routines%
      (upgcnv)   PROCEDURE   % get user program address from name or
```

```
    number%                                                          8B1
       (str);   % address of string containing name or number %   8B1A
       % given the name or number of a user program in a string, look
       up the name in the string of user program names and/or the
       address in the stack of user program starting addresses %
       LOCAL pgmidx, i ,j;
       REF str;
       IF str.L = empty THEN pgmidx _ 0
       ELSE IF *str* [1] IN ['0, '9] THEN
          BEGIN
          pgmidx _ cvsno (&str);
          % check to see if there are any user subsytems loaded, and
          adjust the pgmidx accordingly. User subsystems appear as
          one program to the user but actually take up two entries in
          the upgstk stack%
          j _ 1;
          FOR i_1 UP 1 UNTIL  >= pgmidx DO
             BEGIN
             IF upgstk[j].LH = 777777B THEN BUMP j;
             BUMP j;
             END;
          pgmidx _ j;
          END
       ELSE
          BEGIN
          astruc( &str );
          pgmidx _ upgskix;
          FIND SE(*upgnms*) $NP;
          WHILE NOT (FIND < ENDCHR) DO
             BEGIN
             FIND < $NP $PT;   % move to next visible %
             IF (FIND > *str* (NP/ENDCHR)) THEN EXIT LOOP
             ELSE BUMP DOWN pgmidx;
             IF pgmidx > 1 AND upgstk[ pgmidx-1].LH = 777777B THEN
             BUMP DOWN pgmidx;
             END;
          END;
       IF NOT pgmidx IN [1, upgskix] THEN
          ABORT (upgerr, $"illegal user program spec");
       RETURN (upgstk[pgmidx]);
       END.

%reset%
   %reset PCP interface routine%
      (xpreset) %Execute Reset Command%                              9A1
         PROCEDURE (
            %FORMALS%
               entity REF,              %entity type%
               destptr REF      %destination pointer%
               );
         LOCAL retry1, retry2 REF, destination REF, da REF, save,
         condir, cdrum, rhostn;
         LOCAL STRING filstring[200], badmsg[200];
         LOCAL TEXT POINTER tp1, tp2;
         %----------------------%
         IF &destptr THEN &destination _ ELEM #destptr#[tppair];
```

```
        &da _ lda(ELEM #entity#[wndw]);
        IF NOT nodisplay THEN dpset(dspno, endfil, endfil, endfil);
        CASE ELEM #entity#[cwtype] OF
           = 53 %- content -%: %Content Analysis%
              BEGIN
              da.dacacode _ 0;
              da.davspec.vscapf _ FALSE;
              END;
           = 58: %- buffer -%
              gpbsz($upgbdf);
           ENDCASE
           IF NOT donthelp THEN
              BEGIN
              *badmsg* _ "The type argument was incorrect.  The
              allowed values are Buffer, Case, Content, ";
              *badmsg* _ *badmsg*, "Link, Name, Temporary, and
              Viewspecs.";
              retry1 _ HELP (badarg, $badmsg, cindex);
              REPEAT CASE (retry1);
              END
           ELSE ABORT (badarg, $"The type argument was incorrect.");
        RETURN;
        END.

%run%
   %run PCP interface routine%
      (xprun)              % Run a user program %                    10A1
         PROCEDURE (pnameptr REF);
         LOCAL t2ptr REF, progname REF;
         LOCAL STRING prog[50];
         % move filename to local string %
            IF &pnameptr THEN &progname _ ELEM #pnameptr#[tppair];
            &t2ptr _ &progname + d2sel;
            *prog* _ progname t2ptr;
         % call the program %
            gpexpgm( $prog );
         RETURN;
         END.

   %run core routine%
      (gpexpgm)  PROCEDURE   % does Goto Program Execute a program %
                                                                    10B1
         (str);   % string containing name or number of the user
         program %                                                  10B1A
         % control is transferred to specified program by means of a
         CALL0%
         LOCAL upgaddr;
         REF str;
         upgaddr _ upgcnv (&str);
         IF NOT upgaddr > 0 THEN
            ABORT (upgerr, $"nO address for program");
            %Previously a signal%
         [upgaddr]();
         RETURN;
         END.
```

```
%set%
   (xpset) %Execute Set Command%                                    11A
      PROCEDURE (
         %FORMALS%
            entity REF, %entity type%
            param1 REF,
            param2 REF,
            param3 REF,
            destptr REF %destination pointer%
            );
         LOCAL retry1, retry2 REF, destination REF, csize;
         LOCAL STRING lbufsz[5], badmsg[200];
      %----------------------%
      IF &destptr THEN &destination _ ELEM #destptr#[tppair];
      IF NOT nodisplay THEN dpset(dspno, endfil, endfil, endfil);
      CASE ELEM #entity#[cwtype] OF
         = 53 %- content -%: %Content Analysis%
            CASE ELEM #param1#[cwtype] OF
               = 61 %- to -%: %pattern%
                  BEGIN
                  xcompile( &entity, &param2, 0, 0, 0, 0, &param3 );
                  END;
               = 62 %on%:
                  BEGIN
                  cspupdate _ lda(&param3);
                  cspvs.vscapf _ TRUE;
                  END;
               = 63 %off%:
                  BEGIN
                  cspupdate _ lda(&param3);
                  cspvs.vscapf _ FALSE;
                  END;
               ENDCASE
               IF NOT donthelp THEN
                  BEGIN
                  retry1 _ HELP (badarg, $"The content specification
                  argument was incorrect.  The allowed values are Off,
                  On, or To.", cindex);
                  REPEAT CASE (retry1);
                  END
               ELSE ABORT (badarg, $"The content specification argument
               was incorrect.");
         = 58 %- buffer -%:
            BEGIN
            getpstring(&param1, $lbufsz);
            csize _ VALUE($lbufsz);
            %set the buffer to the new size%
            IF NOT gpbsz(csize) THEN
               ABORT(badbufsize, $"Invalid size for programs buffer.");
            END;
         ENDCASE
         IF NOT donthelp THEN
            BEGIN
            *badmsg* _ "The type argument was incorrect.  The allowed
            values are Buffer, Content, ";
            *badmsg* _ *badmsg*, "External, Link, Name, Private,
```

```
            Public, Temporary, or Viewspecs.";
            retry1 _ HELP (badarg, $badmsg, cindex);
            REPEAT CASE (retry1);
            END
         ELSE ABORT (badarg, $"The type argument was incorrect.");
      RETURN;
      END.

%show%
   %show PCP interface routine%
      (xpshow) %Execute show Command%                                  12A1
         PROCEDURE (
            %FORMALS%
               entity REF,       %entity type%
               window,  %window to be shown status for%
               rtnlist REF);     %return arg%
            LOCAL
               rhostn, retry, dskcnt, destination REF,
               % stuff for show directory %
                  info, % record saying what was requested %
                  gropk,         % record saying how to group things %
                  sortk;         % record saying how to sort things %
            LOCAL LIST pcplist[2];
            LOCAL STRING filstr[200], badmsg[200], statusstr[1000];
         %----------------------%
         CASE ELEM #entity#[cwtype] OF
            = 58 %- status -%:
               BEGIN
               gpstatus ($statusstr, lda(window));
               #rtnlist#[1] _ USE rtnstring($statusstr);
               END;
            ENDCASE ABORT (badarg, $"The type argument was
            incorrect.");
         IF NOT nodisplay THEN dpset(dspno, endfil, endfil, endfil);
         RETURN;
         END.

   %show support routines%
      (gpstatus) PROCEDURE   % does Goto Program Status display %   12B1
         (str, da);   % address of string in which to put stuff %
         %make up a string containing the status of the user program
         stuff%
         REF str, da;
         *str* _
            EOL, EOL, "Stack of compiled programs (first is #1):", EOL,
               "   ", *upgnms*, EOL, EOL;
         *str* _ *str*,
            "Content Analyzer  ", " program for display area:  ";
         upgsstr (da.dacacode, &str);
         *str* _ *str*,
            "Sequence Generator", " program for display area:  ";
         upgsstr (da.dausqcod, &str);
         *str* _ *str*,
            "Sort Key Extractor", " program for display area:  ";
         upgsstr (da.daukeycod, &str);
         *str* _
```

```
        *str*, EOL, EOL,
        "Current buffer size: ", STRING(upgbsz), " pages = ",
        STRING (512*upgbsz), " words.  ", EOL,
        "Room left in buffer:  ",
           STRING (upgbend - upgffbuf + 1),  " words.", EOL;
     RETURN;
     END.

     (upgsstr) PROCEDURE (pgmaddr, str);                          1281N
        LOCAL i,j;
        LOCAL TEXT POINTER tp1, tp2;
        LOCAL STRING lstr[50];
        REF str;
        IF pgmaddr = 0 THEN *lstr* _ "None"
        ELSE
           BEGIN
           *lstr* _ "None";
           j _ 0;
           FOR i _ 1 UP UNTIL > upgskix DO
              BEGIN
              IF upgstk[i] = pgmaddr THEN
                 BEGIN
                 i _ i - j;  %subtract # of subsystems%
                 FIND SF(*upgnms*);
                 FOR i _ i - 1 DOWN UNTIL = 0 DO FIND $NP $PT;
                 FIND $NP ^tp1 $PT ^tp2;
                 *lstr* _ tp1 tp2;
                 EXIT LOOP 1;
                 END;
              IF upgstk[i].LH = 777777B THEN BUMP j;  %subsystem
              (two words)%
              END;
           IF lstr.L = 0 THEN
              IF NOT ddsymget ($lstr, pgmaddr) THEN *lstr* _
              "None";
           END;
        *str* _ *str*, *lstr*, CR, LF;
        RETURN;
        END.

FINISH
(notes)                                                          14
   The Encapsulator sequentializes statemnts for the compiler and
   passes the sequentialized text one page at a time.  A special code
   is affixed by the Encapsulator to the beginning of every statement.
   This code is ignored by the compiler (for compilation purposes) but
   is returned to the Encapsulator if an error was found.  This code is
   the only mechanism by which the Encapsulator can identify the
   erroneous statement and, thus it has to uniquely identify the
   statement.
   A simple use of stids as the identifier may not be unique since an
   stid contains a file numbet which is dymnamically allocated and may
   be reused.  Same with JFNs-- they are dynamically allocated and the
   US may reuse them.
   The proper way to uniquely describe a statement is to encode the RH
   of the stid along with the filename to which it belongs.  In order
```

to avoid the passing long strings between the Encapsulator and the compiler the Encapsulator must build a data structure associating an internal handle with the filename string, passing only the handle. An NLS system table called "..." should assist in building this structure.
The current version of the the Encapsulator does not implement this algorithm but rather uses the file numbers as the file identifier and thus may fail in the rare situation where too many files are involved in a single compilation (refers to INCLUDEs of different files).
Implementing the correct algorithm requires changes in two procedures: cpprefix-- that affixes the prefix to the statements, and rcrdlink-- that decodes that same prefix.

(cxdetach)	<nine, recfil, 08>	PROCEDURE	4
(deldf)	<nine, recfil, 0536>	CONSTANT =67B	3F
(dettfg)	<nine, recfil, 0532>	LOCAL	3C
(gjint)	<nine, recfil, 0535>	CONSTANT =13B	3E
(gpjfn)	<nine, recfil, 0537>	CONSTANT =206B	3G
(pc)	<nine, recfil, 0533>	LOCAL	3B
(recttg)	<nine, recfil, 0531>	LOCAL	3A
(spjfn)	<nine, recfil, 0538>	CONSTANT =207B	3H
(tda)	<nine, recfil, 0534>	REF	3D

```
< NINE, RECFIL.NLS;3, >, 17-Feb-78 16:51 LLG ;;;; C"modeflg"];
FILE recfil   % <ARCSUBSYS>XL10  <RELNINE>recfil % % (arcsubsys,xl10,)
(RELNINE,RECFIL.rel,) %
ALLOW!
%Declarations%
   (rectfg) = 1;                                                          3A
   (pc) = 1;                                                              3B
   (detflg) = 0;                                                          3C
   (tda) REF;                                                             3D
   (gjinf) CONSTANT = 13B;                                                3E
   (deldf) CONSTANT = 67B;                                                3F
   (gpjfn) CONSTANT = 206B;                                               3G
   (spjfn) CONSTANT = 207B;                                               3H
(cxdetach) PROCEDURE ( rhosti, infile, rhosto, outfile );                  4
   LOCAL injfn, outjfn, savraw;
   LOCAL STRING tempstr[100];
   REF infile, outfile, rawchr;
   CASE rhosti OF
      = 1hostn: NULL;
      ENDCASE err( $"remote file manipulations not implemented yet" );
   CASE rhosto OF
      = 1hostn: NULL;
      ENDCASE err($"Remote File Manipulations Not Implemented Yet");
   injfn _ outjfn _ 0;
   *tempstr* _ "NIL:";
   IF NOT &infile THEN &infile _ $tempstr;
   IF NOT &outfile THEN &outfile _ $tempstr;
   INVOKE (filclose);
   IF NOT (injfn _ sgtjfn( gtjoif, &infile, $lit)) THEN
      err( $"Can't GTJFN for input file" );
   IF NOT SKIP !openf( injfn, 7B10+2B5) THEN
      err( $"Can't OPEN input file");
   IF NOT (outjfn _ sgtjfn( gtjoof, &infile, $lit)) THEN
      err( $"Can't GTJFN for output file" );
   IF NOT SKIP !openf( outjfn, 7B10+1B5) THEN
      err( $"Can't OPEN output file");
   IF nlmode = fulldisplay THEN
      BEGIN
      savraw _ &rawchr;
      xsimdev(lda(), devtiex, typewriter); % simulate a TI %
      IF (&rawchr # savraw) AND (savraw = $auxchr) THEN
         auxsav _ &rawchr := $auxchr;
      END;
   !spjfn( 400000B, (injfn * 1B6) + outjfn );
   !dtach();
   DROP (filclose);
   RETURN;

   (filclose) CATCHPHRASE;                                                 4U
      BEGIN
      IF injfn THEN
         BEGIN
         IF NOT SKIP !closf( injfn ) THEN NULL;
         IF NOT SKIP !rljfn( injfn := 0 ) THEN NULL;
         END;
      IF outjfn THEN
```

```
        BEGIN
        IF NOT SKIP !closf( outjfn ) THEN NULL;
        IF NOT SKIP !rljfn( outjfn := 0 ) THEN NULL;
        END;
     CONTINUE;
     END;

END.


FINISH
(nlsutilty) PROC(modeflg);                                          7
   %Do utilty type of things 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 string, stid2, stid1, stid, stiddone, action, rtime, doany,
   fl, pmyjfn, pcap, da, savrub, savmrk, rubflg, newfile, sysmsg;
   LOCAL TEXT POINTER tp1, tp2, tp3, tp4;
   LOCAL STRING wrkstr[100], filnms[50];
   REF string, fl, da;
   tskerrcnt _ 0;
   stid _ stid1 _ 0;
   &da _ 0;
   INVOKE (sigerr, RETURN);
   %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):                                                  7L6B
         BUMP tskerrcnt;
         *filnms* _ EOL, "Directory Connect Fail";
         dismes (1, $filnms);
         RETURN;
         END;
   %high queue this process if being run by a WHEEL or OPERATOR%
      IF (pcap _ enablw()) NOT= -1 THEN
         BEGIN
         R1 _ 485;
         R2 _ 202B;
         !JSYS 243B;
         disablw(pcap);
         END
      ELSE IF tenex < 13200 THEN dismes (1, $"
      Unable to high queue -- not operator or wheel
      ");
   %Get tasks file %
      *filnms* _ "<NLS>TASKS.NLS";
      stid _ orgstid;
      IF (stid.stfile _ open(0, $filnms : newfile)) = 0 THEN
```

```
      BEGIN
      BUMP tskerrcnt;
      err($"Task File open fail");
      END;
%set stid for "TODO" branch%
   CCPOS SF(stid);
   FIND ^tp1;
   *wrkstr* _ "TODO";
   lookup($tp1, $wrkstr, nametyp);
   IF tp1 = endfil THEN
      BEGIN
      dismes (1, $"No ToDo Branch in Tasks File");
      RETURN;
      END;
%set stid for "DONE" branch (create if need to)%
   stid1 _ getsub(stid _ tp1);
   IF stid1 = stid THEN
      BEGIN
      dismes( 2, $"***  TASKS COMPLETED  ***");
      RETURN;
      END;
   *wrkstr* _ "DONE";
   lookup($tp1, $wrkstr, nametyp);
   IF tp1 = endfil THEN
      BEGIN
      &string _ $"(DONE) Tasks Which Are Done";
      FIND SF(*string*) ^tp1 SE(*string*) ^tp2;
      stiddone _ cinssta(stid, levsuc, $tp1, $tp2);
      END
   ELSE stiddone _ tp1;
%setup rubout mechanism for local use%
   rubflg _ FALSE;   % clear rubout flags %
   rubabt _ FALSE;
   IF modeflg # 4 THEN
      BEGIN
      savrub _ rubabt;
      savmrk _ rubmrk;        % save rubout flag address %
      rubmrk _ $rubflg;       % set my own %
      END;
%Set up Primary Output File%
   IF modeflg # 6 THEN
      pmyjfn _ setupop(0, $"U-OUT.TXT", modeflg, -1)
   ELSE pmyjfn _ 101P;        %use primary jfn%
%set to handle signals from compiler%
   DROP (sigerr);
   INVOKE (sigcomp, utstat);
(utstat):                                                    7T
%get and setup a da for compilers to use%
   &da _ newda(); %get da for compile,print,or copy%
   intdafl(&da);
   da.davspec _ defvs1;
   da.davspc2 _ defvs2;
%finally process "TODO" branch performing functions%
   LOOP
      BEGIN
      IF stid1 = stid OR rubflg THEN EXIT;
```

```
      %Now parse command statement%
         CCPOS SF(stid1);
         %Now, find what we are supposed to do%
            action _ IF FIND ["Compile"] THEN 1
               ELSE IF FIND ["Print"] THEN 2
               ELSE IF FIND ["Copy"] THEN 3
               ELSE 0;
            IF actior = 0 THEN err($"Undefined Command");
         %Now get file name%
            IF NOT FIND $NP ^tp1 1$PT ^tp2 THEN
               err($"Illegal Command Statement");
            *filnms* _ tp1 tp2;
         %Set tp1 and tp2 to command%
            FIND SF(stid1) ^tp1 SE(stid1) ^tp2;
      %Now execute%
         CASE action OF
            =1: %compile%
               BEGIN
               IF modeflg # 4 THEN
                  BEGIN
                  %Type out message to controlling tty%
                  *lit* _ EOL, EOL, "Compiling ", *filnms*, 0;
                  !sout(pmyjfn, chbmty+$lit, 0);
                  END;
               runcmp($filnms, &da);
               END;
            =2: %process print requests%
               BEGIN
               IF modeflg # 4 THEN
                  BEGIN
                  %Type out message to controlling tty%
                  *lit* _ EOL, EOL, "Printing ", *filnms*, 0;
                  !sout(pmyjfn, chbmty+$lit, 0);
                  END;
               outptr($filnms, &da, 1, TRUE);
               END;
            =3: %process copy requests%
               BEGIN
               IF modeflg # 4 THEN
                  BEGIN %Type out message to controlling tty%
                  *lit* _ 15B, 12B, "Copying ", *filnms*, 0;
                  !sout(pmyjfn, chbmty+$lit, 0);
                  END;
               copsrc($filnms);
               END;
            ENDCASE err($"Illegal Command");
         %Now fix up command statement and move it to done list%
            *datesr* _ NULL;
            stid2 _ cmovsta(stiddone,levdown, stid1 :=
            getsuc(stid1), FALSE, 0);
            dpset(dspstrc, stid2, stid1, endfil);
            getdat($datesr);  %date/time of complettion%
            *wrkstr* _ "Completed at " , *datesr*;
            FIND SF(*wrkstr*) ^tp1 SE(*wrkstr*) ^tp2;
            stid2 _ cinsst(stid2, levdown, $tp1, $tp2);
            dpset(dspstrc,stid2, endfil, endfil);
```

```
        % recreate display if tasks loaded and display %
          IF nlmode = fulldisplay AND NOT newfile THEN
            BEGIN
            recred();
            END;
      END;
%restore the state of the world%
   rubabt _ savrub;  % restore state of rubout abort %
   rubmrk _ savmrk;
   % delda(&da); %%get rid of dummy da%
   IF newfile THEN closeu(stid.stfile := 0)
   ELSE cupdfil(stid.stfile :=0, newversion, 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
         !gpjfn(4B5);
         R2.RH _ R2.LH;
         !spjfn(4B5);
         IF NOT SKIP !closf(pmyjfn) THEN NULL;
         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 running detatched%
   IF modeflg = 4  THEN
      BEGIN
      clogout();      %go undo group stuff%
      IF NOT SKIP !closf(-1) THEN NULL; %close any openfiles%
      IF NOT SKIP !lgout() THEN NULL; %logout%
      DROP (sigcomp);
      !haltf();
      END
   ELSE
      BEGIN
      dismes (1, $"*** Tasks Completed ***");
      DROP (sigcomp);
      RETURN;
      END;

(sigerr) CATCHPHRASE(:sysmsg);                                    7AA
   CASE SIGNAL OF
      =-5: %We Got here on a badfile, and who knows how%
         IF stid.stfile = bfilno THEN
            BEGIN
            close(stid.stfile  := 0);
            dismes (1, $"Task List File Bad");
            DROP (sigerr);
            TERMINATE; %RETURN%
            END
         ELSE
```

```
            BEGIN
            err($"Bad File");
            CONTINUE;
            END;
        ENDCASE
           BEGIN
           dismes (1, sysmsg);
           dismes (1, $"Fatal Error");
           DROP (sigerr);
           TERMINATE;
           END;

(sigcomp) CATCHPHRASE (:sysmsg);                                        7AC
   BEGIN
   IF &da THEN delda(&da);
   *datesr* _ NULL;
   stid2 _ cmovsta(stiddone,levdown, stid1 := getsuc(stid1), FALSE,
   0);
   dpset(dspstrc, stid2, stid1, endfil);
   getdat($datesr); % time and date of completion %
   *wrkstr* _ *datesr*, "   ", *[sysmsg]*;
   FIND SF(*wrkstr*) ^tp1 SE(*wrkstr*) ^tp2;
   stid2 _ cinsst(stid2, levdown, $tp1, $tp2);
   dpset(dspstrc,stid2, endfil, endfil);
   %Type out message if not detatched%
      IF modeflg # 4 THEN
         BEGIN
         IF nlmode # fulldisplay THEN
            BEGIN
            *lit* _
               EOL,
               "**********************",
               EOL,
               "**********************",
               EOL,
               "    ", *[sysmsg]*, EOL,
               "**********************",
               EOL,
               "**********************", EOL, 0;
            !sout(pmyjfn, chbmty+$lit, 0);
            END
         ELSE % recreate display if tasks is loaded %
            BEGIN
            IF NOT newfile THEN
               BEGIN
               recred();
               END
            ELSE
               BEGIN
               *lit* _ "** ", *[sysmsg]*, " **", 0;
               !sout(pmyjfn, chbmty+$lit, 0);
               END;
            END;
         END;
   TERMINATE;  %GOTO utstat;%
   END;
```

```
END.

(runcmp) PROC (fnmstr, da);                                          8
   LOCAL stid;
   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 statement in e file, to the
   indicated rel-file.  If anything in the first statement does not
   match the accepted syntax ( [percent] $NP COMPILER [percent] < $NP
   $LD > FILE percent) THEn if the first statement is te FILE statement
   of an L10 Program, the file is compiled as L10 to the relfile under
   REL-NLS with the same name as te FILE.
   %
   REF fnmstr, da;
   stid _ orgstid;
   INVOKE (sigrun);
   *fnmstr* _ *fnmstr*, ".NLS",  EOL;
   IF NOT (stid.stfile _ raWopen(&fnmstr, 0)) THEN
      err($"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 rel-file name and compiler%
      CCPOS SF(stid);
      IF NOT FIND [*%] $NP ^z1 $PT ^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%
      da.dacsp _ stid;
      IF (tskerrcnt _ cpcmpfl(TRUE, $cmpnam, $rfilnm, &da)) > 0 THEN
         BEGIN
         *lit* _  STRING(tskerrcnt), " Errors";
         err($lit);
         END;
   %Close source file%
      close(stid.stfile := 0);
   %Return normally%
   DROP (sigrun);
   RETURN;

   (sigrun) CATCHPHRASE;                                             8W
```

```
    BEGIN
    DISABLE (sigrun);
    CASE SIGNAL OF
       =ofilerr:
          BEGIN
          BUMP tskerrcnt;
          err($"Rel-File open fail");
          CONTINUE;
          END;
       =prcerr:
          BEGIN
          BUMP tskerrcnt;
          err($"Compiler open fail or no such compiler");
          CONTINUE;
          END;
       ENDCASE
          BEGIN %error%
          BUMP tskerrcnt;
          IF stid.stfile THEN close(stid.stfile  := 0);
          CONTINUE;
          END;
    END;

END.


(setupop) PROC (jfn, filnamstr, modeflg, tty);                                  9
   %This procedure opens a file with the name in filnamstr if jfn = 0,
   otherwise uses file identified by jfn%
   %It returns the JFN of the new Primary output file%
   %Types out a primary outEut file messae to tty indicated (-1 means
   controling)%
   %Detatches if modeflg = 4%
   LOCAL STRING filnms[50];
   REF filnamstr;
   %Open a sequential file for primary output%
      IF jfn = 0 THEN
         BEGIN
         *filnms* _ *filnamstr*, 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* _ EOL, "Primary Output File is ", *lit*;
            IF tty # -1 THEN
               specttyout(tty, $lit)
            ELSE
               BEGIN
               dismes (1, $lit);
               END;
         R1 _ jfn;
         R2 _ 3;
         IF NOT SKIP !JSYS delnf THEN NULL;
```

```
        END;
    %Now make it the primary output file%
       R1 _ 4B5;
       !gpjfn(); %get primary JFN's%
       R2.RH _ jfn;
       R1 _ 4B5;
       !spjfn(); %set Primary JFN's%
    %Type out detatching message and detach%
       IF (modeflg = 4) AND tty = -1 THEN
          BEGIN
          *filnms* _EOL, "Detaching" , EOL;
          dismes (1, $filnms);
          !dtach();
          END;
    RETURN(jfn);
    END.
(outptr) PROC (fnmstr, da, copies, lpt);                                10
    LOCAL stid, pfjfn;
    LOCAL TEXT POINTER z1, z2;
    LOCAL STRING  pfilnm[50], filnmstring[50]; %For rel-file name%
    % This procedure accepts address of astring containing a file name
    and does an output quickprint on the file.  It sets the extension to
    be the string equivalent of copies.  If lpt is TRUE, then the file
    is put into the printers directory, otherwise in the connected
    directory. %
    REF fnmstr,da;
    stid _ orgstid;
    INVOKE (sigclose);
    *filnmstring* _ *fnmstr*, ".NLS",  EOL;
    IF NOT (stid.stfile _ rawopen ($filnmstring, 0)) THEN
       err ($"Cannot Open File"); %Open Source code file%
    %Get output file name%
       *pfilnm* _ NULL;
       CCPOS SP(*filnmstring*);
       FIND SNP  ^z1 SE(z1) ^z2 >  z1 ([">] ^z1/)  ["."] ^z2 _z2;
       IF lpt THEN
          *pfilnm* _ *prtdir*;
       *pfilnm* _ *pfilnm*,  z1 z2, ".", STRING(copies), EOL;
    %turn statement numbers on (on the right)%
       da.dacsp _ stid;
       da.davspec.vsstnr _ da.davspec.vsstnf _ da.davspec.vssidf _ TRUE;
    %call Output Quickprint%
       coutqui ($pfilnm, &da);
    %Close Source File%
       close(stid.stfile := 0);
    %Return normally%
    DROP (sigclose);
    RETURN;

    (sigclose) CATCHPHRASE;                                             10B
       BEGIN
       DISABLE (sigclose);
       IF stid.stfile THEN close (stid.stfile  := 0);
       IF lptjfn THEN sysclose (lptjfn  := 0);
       CONTINUE;
       END;
```

```
BLP,   16-Aug-78 00:25          < NINE, RECFIL.NLS;3, >  10

   END.


(copsrc) PROC (fnmstr);                                              11
   % This procedure accepts address of astring containing a file name.
   It changes the directory name for the rel file (in the "FILE"
   statement) to NIC-NLS, and then does an output file to the NIC-NLS
   directory. %
   LOCAL stid;
   LOCAL TEXT POINTER ptr1, ptr2, z1, z2;
   LOCAL STRING  nfname[50], lookst[5];
   REF fnmstr;
   stid _ orgstid;
   INVOKE (sigclose);
   *nfname* _ *fnmstr*, ".NLS",  EOL;
   IF NOT (stid.stfile _ rawopen ($nfname, 0)) THEN
      err ($"Cannot Oper File"); %Open Source code file%
   makeptr(stid, $ptr1);
   *lookst* _ "FILE";
   lookup($ptr1, $lookst, contnt);
   IF ptr1 # endfil THEN
      BEGIN %change directory name%
      stid _ ptr1;
      CCPOS SF(stid);
      IF FIND 2[^%] < [^>] ^ptr2 [^<] > CH ^ptr1 THEN
         ST stid _ SF(stid) ptr1, "NIC-NLS", ptr2 SE(stid);
      END;
   %make output file name%
      IF *fnmstr*[1] = ^< THEN
         BEGIN
         CCPOS SF(*fnmstr*);
         IF NOT FIND [^>] ^ptr1 THEN
            err($"Illegal file name");
         *nfname* _ "<NIC-NLS>", ptr1 SE(*fnmstr*), ".NLS", EOL;
         END
      ELSE *nfname* _ "<NIC-NLS>", *fnmstr*, ".NLS", EOL;
   %now do output and clean up%
      cupdfil(stid.stfile, newname, $nfname);
      %Use nfname as work string for transdir%
      *nfname* _ "NIC-NLS";
      FIND SF(*nfname*) ^z1 SE(*nfname*) ^z2;
      csetlindef(lcfile(), $z1, $z2);
      close(stid.stfile := 0);
      DROP (sigclose);
   RETURN;

   (sigclose) CATCHPHRASE;                                           11R
      BEGIN;
      DISABLE (sigclose);
      IF stid.stfile THEN close (stid.stfile  := 0);
      CONTINUE;
      END;

   END.
```

(ccignore)	<nine, sdata, 066>	EXT	3E3
(cfreeblk)	<nine, sdata, 064>	EXT	3D2
(cgetblk)	<nine, sdata, 058>	EXT	3D1
(cinstid)	<nine, sdata, 086>	EXT	5B2
(clink)	<nine, sdata, 060>	EXT	3D6
(cmakezone)	<nine, sdata, 063>	EXT	3D3
(crepienish)	<nine, sdata, 062>	EXT	3D4
(csstk)	<nine, sdata, 083>	EXT	5A3
(csstkb)	<nine, sdata, 084>	EXT	5A2
(csstkx)	<nine, sdata, 081>	EXT	5A1
(cunilink)	<nine, sdata, 061>	EXT	3D5
(entvec)	<nine, sdata, 07>	EXT	3A1
(gmtflag)	<nine, sdata, 096>	EXT	3A8
(hostflag)	<nine, sdata, 094>	EXT	3A7
(ilevel)	<nine, sdata, 087>	EXT	5B3
(ilsadd)	<nine, sdata, 092>	EXT	3B3
(ilsdsp)	<nine, sdata, 012>	EXT	3B1
(ilsins)	<nine, sdata, 091>	EXT	3B2
(ilstxt)	<nine, sdata, 093>	EXT STRING	3B4
(icltimzon)	<nine, sdata, 097>	EXT	3A9
(lhostn)	<nine, sdata, 09>	EXT	3A3
(msgtrk)	<nine, sdata, 080>	EXT	4A3
(msgtck)	<nine, sdata, 078>	EXT	4A1
(msgtim)	<nine, sdata, 079>	EXT	4A2
(nojournalflag)	<nine, sdata, 010>	EXT	3A4
(omode)	<nine, sdata, 085>	EXT	5B1
(p1)	<nine, sdata, 016>	EXT TEXT POINTER	3C1
(p2)	<nine, sdata, 088>	EXT TEXT POINTER	3C2
(psireg)	<nine, sdata, 014>	EXT	3B6
(psirglast)	<nine, sdata, 054>	EXT	3B7
(sav40)	<nine, sdata, 068>	EXT	3F1
(savchntab)	<nine, sdata, 065>	EXT	3E1
(svac1)	<nine, sdata, 071>	EXT	3F2
(svacs)	<nine, sdata, 070>	EXT	3F3
(svacse)	<nine, sdata, 069>	EXT	3F4
(tenex)	<nine, sdata, 08>	EXT	3A2
(timra1)	<nine, sdata, 077>	EXT	3G3
(timra2)	<nine, sdata, 076>	EXT	3G4
(timra3)	<nine, sdata, 075>	EXT	3G5
(timra4)	<nine, sdata, 074>	EXT	3G6
(timrproc)	<nine, sdata, 073>	EXT	3G2
(timrset)	<nine, sdata, 072>	EXT	3G1
(tops20flag)	<nine, sdata, 089>	EXT	3A5
(trpcnt)	<nine, sdata, 067>	EXT	3E2
(userstr)	<nine, sdata, 04>	EXT STRING	2A

```
< NINE, SDATA.NLS.2, >, 23-Jun-77 12:27 KJM ;;;;
FILE sdata   % <ARCSUBSYS>XL10 <RELNINE>sdata   %% (ARCSUBSYS,XL10,)
(RELNINE,sdata.rel,) %

%identification system%
   (userstr) EXTERNAL STRING [50]; %string corresponding to login dir
   name% %PASSED%                                                    2A
% DECLARE's %
   %pre-save variables%
      (entvec) EXTERNAL[77]; %entry vector for nls%                  3A1
      (tenex) EXTERNAL;  %tenex version number%                      3A2
      (lhostn) EXTERNAL;  %logical host number on network%           3A3
      (nojournalflag) EXTERNAL = 0; %disable journal, ident checking%
                                                                     3A4
      (tops20flag) EXTERNAL = 0;  %0 for tenex, 1 for tops20%        3A5
      (nonetworkflag) EXTERNAL = 0; %if nonzero, is "hostnumber" for
      non-network system%
      (hostflag) EXTERNAL = 0;                                       3A7
         %non-zero (=host number) if for broadcasting to system other
         than one on which save file is created%
      (gmtflag) EXTERNAL = 0;  %for tops20, = 1: running GMT as local
      time, ISID glitch%                                             3A8
      (lclTimzon) EXTERNAL = 8;  %constant for Pacific time, but could
      be U.C.%                                                       3A9
   %...illegal instruction pseudo-interrupt support...%
      (ilsdsp) EXTERNAL;  %dispatch address%                         3B1
      (ilsins) EXTERNAL;  %contents of illegal instruction%          3B2
      (ilsadd) EXTERNAL;  %address of illegal instruction%           3B3
      (ilstxt) EXTERNAL STRING [200];  %error message%               3B4
      %DO NOT REORDER THE NEXT TWO DECLARATIONS%
      (psireg) EXTERNAL[15];                                         3B6
      (psirglast) EXTERNAL;  %registers at time of psi%              3B7
   % text pointers %
      (p1) EXTERNAL TEXT POINTER;                                    3C1
      (p2) EXTERNAL TEXT POINTER;                                    3C2
   %...storage management...%
      (cgetblk) EXTERNAL;                                            3D1
      (cfreeblk) EXTERNAL;                                           3D2
      (cmakezone) EXTERNAL;                                          3D3
      (creplenish) EXTERNAL;                                         3D4
      (cunlink) EXTERNAL;                                            3D5
      (clink) EXTERNAL;                                              3D6
   %...pseudointerrupt tables... moved to (nine,dpsvjs,)%
   %PASSED%
      (savchntab) EXTERNAL[4]; %for use in no-oping first four pseudo
      interrupts%                                                    3E1
      (trpcnt) EXTERNAL = 0;                                         3E2
      (ccignore) EXTERNAL;  %count of how many times to ignore control
      characters%                                                    3E3
   %...pseudointerrupt data...% %PASSED%
      (sav40) EXTERNAL;                                              3F1
      (svac1) EXTERNAL;                                              3F2
      (svacs) EXTERNAL[15];                                          3F3
      (svacse) EXTERNAL;                                             3F4
   %...system timer globals...%
      (timrset) EXTERNAL=0;   %timer is running flag%                3G1
```

```
      (timrproc) EXTERNAL;  %procedure to be called when the timmer
      goes off%                                                          3G2
      (timra1) EXTERNAL;  % argument 1 for timrproc %                    3G3
      (timra2) EXTERNAL;  % argument 2 for timrproc %                    3G4
      (timra3) EXTERNAL;  % argument 3 for timrproc %                    3G5
      (timra4) EXTERNAL;  % argulent 4 for timrproc %                    3G6
%...global display/print parameters...%
   %...display support...%
      (msglck) EXTERNAL;                                                 4A1
      (msgtim) EXTERNAL;                                                 4A2
      (msgfrk) EXTERNAL;                                                 4A3
% for use with syntax generating commands % % this could be moved to
the syntax generating subsystem %
   % stack and stack pointers for partial path generation %
      (csstkx) EXTERNAL;                                                 5A1
      (csstkb) EXTERNAL;                                                 5A2
      (csstk) EXTERNAL[100];                                             5A3
   % variables controlling output mode %
      (omode) EXTERNAL = 0;     % output modes %                         5B1
         % 0 - debugging: do typeas in onode %
         % 1 - inserting: do insert statement in onode %
            % cinstid = stid of statement to insert after %
            % ilevel = level to insert at %
         % 2 - being used as seggen: do send in onode %
            % instid = adr of seg work area %
         % 3 - being used in show type command: call fbctl in onode %
      (cinstid) EXTERNAL;                                                5B2
      (ilevel) EXTERNAL;                                                 5B3

FINISH of sdata
```

(appseq)	<nine, seqfil, 01953>	PROCEDURE	9A
(ascmbk)	<nine, seqfil, 010>	CONSTANT =175B	4L
(bldsfhdr)	<nine, seqfil, 0570>	PROCEDURE	5L
(cat)	<nine, seqfil, 013>	CONSTANT =0	4N
(entering)	<nine, seqfil, 016>	CONSTANT =0	4Q
(fhand)	<nine, seqfil, 01505>	FIELD - 26	8B1
(formfeed)	<nine, seqfil, 017>	STRING	4F
(getsbn)	<nine, seqfil, 01661>	PROCEDURE	8Z
(niop)	<nine, seqfil, 01222>	PROCEDURE	6E
(nmapin)	<nine, seqfil, 01895>	PROC	8G
(nmapout)	<nine, seqfil, 01507>	PROC	8C
(nndacc)	<nine, seqfil, 01506>	FIELD - 10	8B2
(inseq)	<nine, seqfil, 0610>	PROCEDURE	6A
(inseqi)	<nine, seqfil, 01165>	PROCEDURE	6D
(inseqn)	<nine, seqfil, 01051>	PROCEDURE	6C
(isread)	<nine, seqfil, 01012>	LOCAL	6B
(leaving)	<nine, seqfil, 015>	CONSTANT =-1	4P
(mapin)	<nine, seqfil, 01534>	PROC	8F
(mapout)	<nine, seqfil, 01919>	PROC	8D
(mapped)	<nine, seqfil, 01504>	RECORD	8B
(maxlev)	<nine, seqfil, 07>	CONSTANT =63	4C
(maxmclv)	<nine, seqfil, 01845>	CONSTANT =1	4K
(maxtrc)	<nine, seqfil, 01844>	CONSTANT =63	4J
(n40fst)	<nine, seqfil, 01840>	CONSTANT =40B	4D
(ok)	<nine, seqfil, 012>	CONSTANT =1	4M
(opinit)	<nine, seqfil, 01644>	PROCEDURE	8Y
(oprchr)	<nine, seqfil, 01570>	PROCEDURE	8I
(oprcid)	<nine, seqfil, 01808>	PROCEDURE	8U
(oprdc)	<nine, seqfil, 01795>	PROCEDURE	8T
(oprdt)	<nine, seqfil, 01825>	PROCEDURE	8W
(oprdtc)	<nine, seqfil, 01782>	PROCEDURE	8S
(oprgps)	<nine, seqfil, 01612>	PROCEDURE	8L
(oprhdf)	<nine, seqfil, 01640>	PROCEDURE	8Q
(oprint)	<nine, seqfil, 01817>	PROCEDURE	8V
(oprlev)	<nine, seqfil, 01624>	PROCEDURE	8N
(oprnst)	<nine, seqfil, 01576>	PROCEDURE	8J
(oprrst)	<nine, seqfil, 01616>	PROCEDURE	8M
(oprsid)	<nine, seqfil, 01778>	PROCEDURE	8R
(oprsig)	<nine, seqfil, 01634>	PROCEDURE	8P
(oprsvc)	<nine, seqfil, 01629>	PROCEDURE	8O
(oprtxt)	<nine, seqfil, 01599>	PROCEDURE	8K
(opseqf)	<nine, seqfil, 0144>	PROCEDURE	5F
(opssig)	<nine, seqfil, 02012>	PROC	5G
(optypefe)	<nine, seqfil, 02059>	PROCEDURE	8X
(outasm)	<nine, seqfil, 0134>	PROCEDURE	5E
(outjm)	<nine, seqfil, 038>	PROCEDURE	5C
(outqp)	<nine, seqfil, 029>	PROCEDURE	5B
(outseq)	<nine, seqfil, 019>	PROCEDURE	5A
(pager)	<nine, seqfil, 0119>	PROCEDURE	5D
(pmfmap)	<nine, seqfil, 01873>	PROCEDURE	8E
(prgetst)	<nine, seqfil, 01556>	PROCEDURE	8H
(processor)	<nine, seqfil, 01365>	PROCEDURE	8A
(seqeof)	<nine, seqfil, 01843>	CONSTANT =1B9	4I
(seqer)	<nine, seqfil, 09>	CONSTANT =4B8	4H
(sfhfnm)	<nine, seqfil, 08>	CONSTANT =48	4F
(sfhpgn)	<nine, seqfil, 01842>	CONSTANT =70	4G

(sfmacpt)	<nine, seqfil, 0462>	PROCEDURE	5J
(sfnxtst)	<nine, seqfil, 0265>	PROCEDURE	5H
(sfputst)	<nine, seqfil, 0278>	PROCEDURE	5I
(sttrin)	<nine, seqfil, 0524>	PROCEDURE	5K
(submission)	<nine, seqfil, 014>	CONSTANT =1	40
(tenfst)	<nine, seqfil, 01841>	CONSTANT =0	4E

ue%%% 000005 IH@ SAA@@AA@(

< NINE, SEQFIL.NLS;17, >, 3-Jun-78 00:04 ROM ;;;;

FILE seqfil % <ARCSUBSYS>L109 to <RELNINE>SEQFIL % % (arcsubsys,L109,)
(RELNINE,seqfil.rel,) %

ALLOW!
%....compile-time switches....%
   SET NSW = FALSE; %TRUE => compile to (RELNINE,wmseqfil.rel,) %
%....declarations.....%

```
    p = 7,
    wa = 8;
REF prseqwk, tda;
(maxlev) CONSTANT = 63;                                                    4C
(n40fst) CONSTANT = 40B;                                                   4D
(tenfst) CONSTANT = 0;                                                     4E
(sfhfnm) CONSTANT = 48;                                                    4F
(sfhpgn) CONSTANT = 70;                                                    4G
(seqer) CONSTANT = 4B8;                                                    4H
(seqeof) CONSTANT = 189;                                                   4I
(maxtrc) CONSTANT = 63;                                                    4J
(maxmclv) CONSTANT = 1;                                                    4K
(ascmbk) CONSTANT = 1758;                                                  4L
(ok) CONSTANT = 1;           % successful return %                        4M
(cat) CONSTANT = 0;          % catastrophic error return %                4N
(submission) CONSTANT = 1; % journal submission %                          4O
(leaving) CONSTANT = -1;   % file leaving system %                         4P
(entering) CONSTANT = 0;   % file entering system %                        4Q
(formfeed) STRING = "
```

```
   ";                                                                    4R
%.....output sequential, quickprint, and assembler.....%

   (outseq) PROCEDURE   % does the Output Sequential File command %     5A
      (ofilnam,   % string containing name of the output file %
      da,         % display area descriptor %
      forceup);   % whether alphas are to be forced upper case %
      REF ofilnam, da;
      dismes(1, $"Output Sequential in Progress");
      opseqf (&ofilnam, &da, forceup, FALSE, opsqfl);
      dismes(0);
      RETURN;
      END.

   (outqp)  PROCEDURE   % does the Output Quickprint command %          5E
      (ofilnam,   % string containing name of the output file %
      da);        % display area descriptor %
      REF ofilnam, da;
      dismes (1, $"Output Quickprint in Progress");
      opseqf (&ofilnam, &da, FALSE, TRUE, opqpfl);
      dismes(0);
      RETURN;
      END.

   (outjm) PROCEDURE ( str, da ); % does Output Journal Mail command %
                                                                        5C
      % FORMAL PARAMETERS %
         % str  - address of astring containing output file name %
         % address of display area %
      LOCAL pjb, pjba, pjbb, pjbc;
         %may delete this statement and make pjb a formal parameter%
      LOCAL pjl, pjbrad, pjfile;
      LOCAL pjbflag %journal branch found%, pjoflag %mail found%;
      LOCAL pjcsp, pjrubmrk, pjcacode;
      LOCAL TEXT POINTER pjtp, tp1, tp2;
      REF str, da, pjb;
      % set branch names, etc. %
         pjbflag _ pjoflag _ 0;
         %may delete these when pjb is made a calling parameter%
            &pjb _ $pjba;
            pjba_$"Journal"; pjbb_$"Info"; pjbc_$"Action";
         pjl_3;
         pjfile _ da.dacsp.stfile;
      WHILE (pjl_pjl-1) >= 0 DO IF (pjbrad _ pjb[pjl]) THEN
         BEGIN
         % fetch stid for head of Journal branch %
            b1 _ origin; % build stid %
            b1.stfile _ pjfile; % for origin statemnt %
            b1[1] _ 1; % of currently loaded file %
            b1 _ namelook(b1, pjbrad); % fetch stid for Journal,
            Action, or Info branch %
            IF b1 = endfil THEN REPEAT LOOP; % there may not be any %
            pjbflag _ TRUE;
            IF getsub(b1) = b1 THEN REPEAT LOOP; %no mail here%
            pjtp _ b1; % save stid for head %
            pjtp[1] _ b1[1]; % of Journal branch %
```

```
    IF NOT pjoflag THEN
       BEGIN
       % save away current display area csp and viewspecs %
          pjusqc _ da.dausqcod; % save user seq gen addr %
          pjsavf _ da.davspec; % and viewspecs %
          pjcsp _ da.dacsp; % save csp for restoration %
          pjrubmrk _ rubmrk; % save pointer to curr rubout flag %
          pjcacode _ da.dacacode;
       % trap signals so can restore the display area stuff %
          INVOKE (rstrda); % can't afford to lose control %
       % set up display area stuff for new sequence %
          da.dausqcod _ $pjseqg; % instate Print Journal seq gen %
          da.davspec.vsusqf _ TRUE; % and enable it %
          da.davspec.vsbrof _ TRUE; % branch only %
          da.dacacode _ $pager; % content analyzer to put out form
          feeds %
          da.davspec.vscapf _ TRUE; % turn on content analyzer %
          rubmrk _ $pjrubout; % intercept RUBOUT %
       % set flag so this won't be done again %
          pjoflag _ TRUE;
       dismes(1, $"Output of Journal Mail in progress" );
       END;
    % output the file %
       da.dacsp _ b1;
       opseqf( &str, &da, FALSE, TRUE, opqpfl );
    END;
 IF NOT pjbflag THEN err($" No Journal Branches");
 IF pjoflag THEN
    BEGIN
    % restore the saved display area state stuff %
       dismes(0);
       da.davspec _ pjsavf; % restore user seq gen addr %
       da.dausqcod _ pjusqc; % and  viewspecs %
       da.dacsp _ pjcsp; % restore csp %
       da.dacacode _ pjcacode;
       rubmrk _ pjrubmrk; % restore rubout flag pointer %
    END
 ELSE err($" You have no Journal mail");
 DROP (rstrda);
 RETURN; % return to caller %

 (rstrda) CATCHPHRASE;                                          5C16
    BEGIN
    CASE SIGNALTYPE OF
       = aborttype:
       BEGIN
          da.davspec _ pjsavf; % w/o restore user seq gen addr %
          da.dausqcod _ pjusqc; % and viewspecs %
          da.dacsp _ pjcsp; % restore csp %
          da.dacacode _ pjcacode;
          rubmrk _ pjrubmrk; % restore rubout flag pointer %
       END;
    ENDCASE;
    CONTINUE;
    END;
```

```
    END.

(pager) PROCEDURE ( sw );                                          5D
    % FORMAL PARAMETERS %
       % sw   address of the sequence generator work area %
    % TEXT POINTERS %
       LOCAL TEXT POINTER  tp1;   % local copy of CCPOS %
    REF sw;
    % set tp1 to be the current CCPOS -- do not use CCPOS as it might
    be used by some routine downstream from us %
       FIND ^tp1;
    % look for a statment that looks like a journal citation %
       IF FIND tp1 f"Location:"] THEN
          BEGIN % journal citation, eject to a new page %
          send(&sw,$formfeed);
          END;
    RETURN(TRUE);
    END.

(outasm) PROCEDURE   % does the Output to Assembler File command % 5E
    (ofilnam,    % string containing name of the output file %
    da,          % display area descriptor %
    forceup);    % whether alphas are to be forced upper case %
    REF ofilnam, da;
    dismes(1, $"Output for Assembler in Progress");
    opseqf (&ofilnam, &da, forceup, FALSE, opmcfl);
    dismes(0);
    RETURN;
    END.

(opseqf) PROCEDURE                                                 5F
    (ofilnam,    % string containing name of the output file %
    da,          % display area descriptor %
    upflg,       % whether to force alphabetic characters uppercase %
    pgflg,       % whether to paginate %
    typfil);     % type of file %

    %This is the main control file for generating a sequential file.
    Basically, it uses the sequence generator to get new statements
    and then adds these statements to a sequential file.  The file is
    both opened and closed here. %
    %------------------%
    LOCAL jfn, sw, toplev, seqstd, sv1, sv2, opsstate, lptflag;
    LOCAL STRING string[200];
    REF ofilnam, da, sw;

    % open the output file %
       INVOKE (sigops);
       opsstate _ 0;
       IF (NOT oqapfg) OR ( FIND SF(*ofilnam*) "< *prtdir* "> ) THEN
          toplev _ gtjoosf
       ELSE toplev _ 0;
       lptflag _ IF (FIND SF(*ofilnam*) "LPT:") THEN TRUE ELSE FALSE;
       IF NOT jfn _
          lgetjfn (0, &ofilnam, $txtext, toplev, $lit) THEN
             ABORT (ofilerr, $lit);
```

```
      opsstate _ 1;
      IF lptflag THEN oqapfg _ FALSE
      ELSE
         BEGIN
         !gtfdb( jfn, 136+1, $R3);
         IF oqapfg THEN oqapfg _ IF R3.fdbnxf THEN FALSE ELSE TRUE;
         END;
      IF NOT sysopen (jfn, IF oqapfg THEN append ELSE write, chrtyp,
      $lit) THEN
         BEGIN
         reljfn (jfn);
         ABORT (ofilerr, $lit);
         END;
      opsstate _ 2;
   %initialize formatting stuff%
      sfucf  _ upflg;
      sfpgno _ pgflg;
      sflnel _ CASE typfil OF
         =opqpfl: IF uqpcolmax=0 THEN defqcolmax ELSE uqpcolmax;
         ENDCASE 72;
      sfndlv _ da.daind/hinc;
      sfmxnd _ da.damind/hinc;
   %initialize sequence generator%
      &sw _ openseq (seqstd _ da.dacsp, seqend(seqstd, sv1 _
      da.davspec, sv2 _ da.davspc2), sv1, sv2, da.dausqcod,
      da.dacacode);
   opsstate _ 3;
   %initialize stuff for doing pmap's%
      sfpmr1.LH _ 485;
      sfpmr1.RH _ $sfbuff/1000B;
      sfpmr2.LH _ jfn;
      sfpmr2.RH _ 0;
      sfpmr3 _ 140001B6;
      sfbyte _ 0;
      sfbufl _ slngth(
         (sfbptr _ stbptr(empty) + $sfbuff),
         (sfndbf _ stbptr(5) + $sfbufe) );
   %initialize for output quickprint append%
   IF oqapfg AND (typfil = opqpfl) THEN
      BEGIN
      R1 _ jfn;
      R2 _ 'L - 1000;
      !JSYS bout;
      END;
   opsstate _ 4;
   IF (IF typfil=opqpfl THEN &sw ELSE sfnxtst(&sw)) THEN
      BEGIN
      IF sfpgno THEN
         BEGIN
         bldsfhdr(&da);
         *string* _ CR, LF, *sfhead*, '1, CR, LF, LF;
         sflnpg _ 3;
         CCPOS SF(*string*);
         sftrln(stbptr(empty)+$string+1,
         stbptr(string.L)+$string+1);
         BUMP sfpgno;
```

```
        END;
      CASE typfil OF
        = opmcfl:
          BEGIN
          IF (toplev _ sw.swclvl) = 0 THEN toplev _ 1;
          sfmacpt(&da, &sw, toplev);
          END;
        = opqpfl: NULL;
        ENDCASE sfputst(&da, &sw, typfil);
      WHILE sfnxtst(&sw) DO
        IF typfil=opmcfl THEN sfmacpt(&da, &sw, toplev)
        ELSE sfputst(&da, &sw, typfil);
      END;
    % close sequence %
      closeseq (&sw);
    %close file%
      IF NOT oqapfg THEN
        BEGIN
        %pmap last page out%
          R1 _ sfpmr1;
          R2 _ sfpmr2;
          R3 _ sfpmr3;
          !JSYS pmap;
        END
      ELSE
        BEGIN
        %sout last page out%
          R1_sfpmr2.LH;
          R2.RH_sfpmr1.RH*1000B;
          R2.LH_-1;
          R3_0;
          !JSYS sout;
        END;
      opsstate _ 3;
      %dismiss page from fork%
        R1 _ -1;
        R2 _ sfpmr1;
        R3 _ sfpmr3;
        !JSYS pmap;
      opsstate _ 2;
      sfbyte _ sfbyte + slngth((stbptr(empty) + $sfbuff), sfbptr);
      R1.LH _ 12B;
      R1.RH _ jfn;
      R2 _ -1;
      R3 _ sfbyte;
      !JSYS chfdb;
      % close output file (but keep jfn) %
        IF NOT sysclose (jfn .V 4B11, $lit) THEN
          BEGIN
          reljfn (jfn);
          dismes (2, $lit);
          ABORT( statesig, 0);
          END;
      opsstate _ 1;
      % delete old versions, if not Output Quickprint or Mailfile %
        IF typfil # opqpfl AND typfil # opmlfl THEN delovsrns (jfn,
```

```
        nvtk);
      opsstate _ 0;
      % release the jfn %
        reljfn (jfn);
   RETURN;

   (sigops) CATCHPHRASE;                                          5F25
      BEGIN
      DISABLE (sigops);
      CASE SIGNALTYPE OF
         = aborttype:
            opssig($opsstate, &sw, jfn);
         ENDCASE;
      CONTINUE;
      END;
   END.

(opssig) PROC(stflag, sw, jfn);                                   5G
   %signal routine for opseqf, close down stuff depending on stflag
   value.  the greater the value, the more stuff to do%
   REF stflag, sw;
   LOOP CASE (stflag := stflag - 1) OF
      =4: !pmap(-1, sfpmr1, sfpmr3); %map work page out of addr.
      space%
      =3: closeseq(&sw);
      =2:
         BEGIN %close file and release jfn%
         sysclose(jfn, $lit);
         RETURN;
         END;
      =1:
         BEGIN
         reljfn(jfn);
         RETURN;
         END;
      ENDCASE RETURN;
   END.

(sfnxtst) %***% PROCEDURE (sw);                                   5H
   %This procedure calls the sequence generator to find the next
   statement.  If there are no more statements in the sequence, it
   returns FALSE.  Otherwise, it initializes CCPOS for reading the
   next statement, which may involve skipping the statement name. %
   %------------------%
   LOCAL TEXT POINTER tptr;
   REF sw;
   IF inptrf OR (tptr _ seqgen(&sw)) = endfil THEN
      RETURN(FALSE);
   tptr[1] _ IF sw.swvspec.vsnamf THEN 1
      ELSE fchtxt(tptr);
   CCPOS tptr;
   RETURN(TRUE);
   END.

(sfputst) PROCEDURE (da, sw, typfil);                             5I
   %This copies the contents of READC to a local string. It expects
```

```
    READC to be set up to begin reading.  When it completes
    processing this statement it performs the necessary cleanup to
    begin processing the next statement (by calling SFENDST) and
    returns.%
    %------------------%
    LOCAL TEXT POINTER tptr;
    LOCAL curlin, chrcnt, indcnt, begbp, bytptr, char, tabpos, count,
    inblnk, outblnk, frstblnk, cntrlstr, mrgnlvl, extraline, sigend,
    stnobp, prevchar, string[50];
    LOCAL STRING stnsig[30];
    REF da, sw;
    %initialize this statement%
       IF sw.swvspec.vsindf OR sw.swvspec.vsrind THEN
          BEGIN
          IF sw.swvspec.vsrind AND (sw.swvspec.vsplxf OR
          sw.swvspec.vsbrof) THEN mrgnlvl _ sw.swslvl
             ELSE mrgnlvl _ 1;
          indcnt _ MAX (0, MIN (sfndlv * (sw.swclvl-mrgnlvl),
          sfmxnd));
          END
       ELSE indcnt _ 0;
       bytptr _ begbp _ stbptr(empty) + $string;
       IF (chrcnt _ indcnt) > 0 THEN
          FOR count _ 1 UP UNTIL > indcnt DO ^bytptr _ SP;
       IF sw.swvspec.vsstnf AND NOT sw.swvspec.vsstnr
       AND sw.swcstid.stpsid # origin AND NOT sw.swstid.stastr THEN
          BEGIN % statement number on left %
          *stnsig* _ NULL;
          IF sw.swvspec.vssidf
             THEN % display sid's %
                *stnsig* _ '0, STRING( getsid( sw.swcstid ) )
             ELSE % display line numbers%
                fechnm (sw.swsvw, $stnsig);
          FOR count _ empty + 1 UP UNTIL > stnsig.L DO
             ^bytptr _ *stnsig*[count];
          ^bytptr _ SP;
          chrcnt _ chrcnt + stnsig.L + 1;
          END;
       curlin _ 1;
       extraline _ FALSE;
    LOOP   % for each line %
       BEGIN
       inblnk _ CCPOS;
       outblnk _ frstblnk _ bytptr;
       LOOP
          CASE char _ READC OF
             = SP:
                BEGIN
                IF (chrcnt _ chrcnt + 1) >= stlnel THEN EXIT LOOP 1;
                ^bytptr _ SP;
                outblnk _ bytptr;
                inblnk _ CCPOS;
                prevchar _ CCPOS;
                END;
             = ENDCHR: EXIT LOOP 2;
             = TAB:
```

```
            BEGIN
            tabpos _ MIN (sflnel,
               fndtab(&da,chrcnt + (IF typfil = opqpfl THEN 0
               ELSE 1)) - 1);
            FOR chrcnt UP UNTIL >= tabpos DO
               ^bytptr _ SP;
            IF chrcnt > sflnel THEN EXIT LOOP 1;
            outblnk _ bytptr;
            inblnk  _ CCPOS;
            END;
         = CR, =EOL:  EXIT LOOP 1;
         < 40B:  %handle control chars for output quickprint%
            IF typfil NOT= opqpfl THEN GO TO eccntrl
            ELSE  %put in printing strings for control
            characters%
               BEGIN

               %force page eject on ^L & don't print anything%
               IF (char = 'L - 100B) AND sfpgno THEN
                  BEGIN
                  sflnpg _ 200; % checked in sftrlu at end of
                  line %
                  GO TO eccntrl;
                  END;

               %get printing string and put in output string%
               cntrlstr _ npstrad(char);
               IF chrcnt + [cntrlstr].L <= sflnel THEN
                  BEGIN   %stash away printing string if it fits%
                  FOR count _ empty + 1 UP UNTIL > [cntrlstr].L
                  DO
                     BEGIN
                     ^bytptr _ *[cntrlstr]*[count];
                     chrcnt _ chrcnt + 1;
                     END;
                  chrcnt _ chrcnt - 1;
                  END
               ELSE chrcnt _ chrcnt + [cntrlstr].L - 1;

               %continue as if normal characters%
               GO TO eccntrl;
               END;
         ENDCASE
            BEGIN
            (eccntrl):                                          5I8D1F2
               IF (chrcnt _ chrcnt + 1) >= sflnel THEN
                  BEGIN
                  IF frstblnk # outblnk THEN %there are blanks%
                     BEGIN
                     bytptr _ outblnk;
                     %reset readwk%
                        tptr _ sw.swstid;
                        tptr[1] _ inblnk;
                        CCPOS tptr;
                     END
                  ELSE    %no blanks; back up 1 char%
```

```
                        BEGIN
                        tptr _ sw.swstid;
                        tptr[1] _ prevchar;
                        CCPOS tptr;
                        END;
                     EXIT LOOP 1;
                     END;
                IF char > 40B THEN
                   BEGIN
                   ^bytptr _ IF sfucf AND char IN ['a, 'z] THEN
                   char - 40B ELSE char;
                   END
                ELSE IF typfil # opgpfl THEN ^bytptr _ char;
                prevchar _ CCPOS;
                END;
    IF curlin >= sw.swvspec.vstrnc THEN EXIT LOOP 1;
    ^bytptr _  CR;
    ^bytptr _  LF;
    sftrln (begbp, bytptr);   % transfer line to output buffer %
    % intialize for next line %
       BUMP curlin;
       bytptr _ begbp;
       IF (chrcnt _ indcnt) > 0 THEN
          FOR count _ 1 UP UNTIL > indcnt DO ^bytptr _ SP;
    END;
% end of statement stuff %
IF sw.swvspec.vsstnf AND sw.swvspec.vsstnr
AND sw.swcstid.stpsid # origin THEN
    BEGIN  %statement number on right%
    *stnsig* _ NULL;
    IF sw.swvspec.vssidf
       THEN % display sid's %
          *stnsig* _ '0, STRING( getsid( sw.swcstid ) )
       ELSE % display line numbers%
          fechnm (sw.swsvw, $stnsig);
    IF chrcnt + 1 + stnsig.L > sflnel THEN
       BEGIN %must go to new line%
       ^bytptr _ CR;
       ^bytptr _ LF;
       sftrln (begbp, bytptr);   % transfer line to output buffer
       %
       bytptr _ begbp;
       chrcnt _ 0;
       extraline _ TRUE;
       END
    ELSE extraline _ FALSE;
    FOR count _ chrcnt + 1 UP UNTIL > sflnel - stnsig.L DO
       ^bytptr _ SP;
    FOR count _ empty + 1 UP UNTIL > stnsig.L DO
       ^bytptr _ *stnsig*[count];
    END;
IF NOT (sw.swvspec.vsidtf AND extraline) THEN
    BEGIN  %only if signatures on and go on separate line%
    ^bytptr _ CR;
    ^bytptr _ LF;
    sftrln (begbp, bytptr);   % transfer line to output buffer %
```

```
      sigend _ sflnel;
      END
   ELSE
      BEGIN
      sigend _ sflnel - stnsig.L - 3;
      stnobp _ bytptr;
      END;
   IF sw.swvspec.vsidtf THEN
      BEGIN
      bytptr _ begbp;
      *stnsig* _ NULL;
      fechsig (sw.swcstid, $stnsig);
      FOR count _ 1 UP UNTIL > sigend - stnsig.L DO
         ^bytptr _ SP;
      FOR count _ empty + 1 UP UNTIL > stnsig.L DO
         ^bytptr _ *stnsig*[count];
      IF extraline THEN bytptr _ stnobp;
      ^bytptr _ CR;
      ^bytptr _ LF;
      sftrln (begbp, bytptr);   % transfer line to output buffer %
      END
   ELSE IF sw.swvspec.vsblkf AND NOT extraline THEN
      BEGIN
      bytptr _ begbp;
      ^bytptr _ CR;
      ^bytptr _ LF;
      sftrln (begbp, bytptr);   % transfer line to output buffer %
      END;
   RETURN;
   END.

(stmacpt) PROCEDURE (da, sw, toplev);                                  5J
   %This copies the contents of READC to a local string, for output
   to the assembler. It works like sfputst, except it inserts tabs
   at the beginning of statements that are of a lower level than the
   first statement in the series. In addition a statement consisting
   of only one space will be output as a null line. %
   %------------------%

   LOCAL char, bytptr, count, begptr, tmpptr, string[400];
   REF da, sw;

   % initialization %
   begptr _ bytptr _ tmpptr _ stbptr(empty) + $string;

   % put out semi-colon in front of origin statement %
   IF (NOT sw.swstid.stastr) AND (sw.swstid.stpsid = origin) THEN
      ^bytptr _ "; ;

   %  put out leading tabs in front of first line of statement if
   appropriate and indenting is turned on  %
   IF (count _ sw.swclvl - toplev) < 0  THEN count _ 0;
   IF count > 0 AND da.davspec.vsindf THEN
      DO ^bytptr _ TAB  UNTIL (count _ count - 1) <= 0;
   tmpptr _ bytptr;
```

```
   LOOP
      BEGIN
      LOOP
         CASE  char _ READC  OF
            = ENDCHR:  EXIT LOOP 1;
            = CR, =EOL:
               BEGIN
               ^bytptr _ CR;
               ^bytptr _ LF;
               EXIT LOOP 1;
               END;
            ENDCASE
               BEGIN
               % convert to upper case if appropriate %
               ^bytptr _ IF sfucf AND char IN ['a, 'z] THEN
                  char - 40B ELSE char;
               END;

      % put out null statement if appropriate %
      IF char = ENDCHR AND ^tmpptr = SP AND slngth(tmpptr,bytptr) =
      0
         THEN bytptr _ begptr;

      CASE char OF
         = ENDCHR: EXIT LOOP 1;
         = EOL, = CR: NULL;
         ENDCASE
            BEGIN
            ^bytptr _ CR;
            ^bytptr _ LF;
            END;
      sftrln(begptr, bytptr);  % transfer line to output buffer %

      % reinitialize for going thru loop %
      begptr _ tmpptr _ bytptr _stbptr(empty) + $string;
      END;

   ^bytptr _ CR;
   ^bytptr _ LF;
   sftrln(begptr, bytptr);  % transfer line to output buffer %
   RETURN;
   END.

(sftrln) PROCEDURE (begbp, bytptr);                                    5K
   %This routine copies characters bounded by begbp and bytptr (both
   byte pointers) into sfbuff and pmaps sfbuff into the print file
   when the buffer is full.%
   %------------------%
   LOCAL STRING string[100];
   UNTIL begbp = bytptr DO
      BEGIN
      ^sfbptr _ ^begbp;
      IF sfbptr = sfndbf THEN
         BEGIN
         IF NOT ogapfg THEN
            BEGIN
```

```
            %pmap page out%
               R1 _ sfpmr1;
               R2 _ sfpmr2;
               R3 _ sfpmr3;
               !JSYS pmap;
            END
         ELSE
            BEGIN
            %sout page out%
               R1_sfpmr2.LH;
               R2.RH_sfpmr1.RH*1000B;
               R2.LH_-1;
               R3_-5000B;
               !JSYS sout;
            END;
         %dismiss page from fork%
            R1 _ -1;
            R2 _ sfpmr1;
            R3 _ sfpmr3;
            !JSYS pmap;
         sfbptr _ stbptr(empty) + $sfbuff;
         BUMP sfpmr2;
         sfbyte _ sfbyte + sfbufl;
         END;
      END;
   IF sfpgno AND (sflnpg _ sflnpg + 1) >= 60 THEN
      BEGIN
      *string* _ 14B, CR, LF, *sfhead*, STRING(sfpgno), CR, LF, LF;
      sflnpg _ 3;
      sftrln (stbptr(empty)+$string+1, stbptr(string.L) +
      $string+1);
      BUMP sfpgno;
      END;
   RETURN;
   END.

(bldsfhdr) PROCEDURE (da);                                          5L
   %This routine builds a page header for output quickprint.%
   %------------------%
   LOCAL vspwrd, count;
   LOCAL STRING filstr[100];
   REF da;
   IF ognhfg THEN
      BEGIN
      %No header wanted; page numbers only%
         *sfhead* _ "
                    Page ";              %57 spaces%
         RETURN;
      END;
   *filstr* _ NULL;
   vspwrd _ da.davspec;
   %put initials into sfhead%
      *sfhead* _ "   ", *initsr*, ",   ";
   getdat($sfhead);
   %put truncation and level values into sfhead%
      IF vspwrd.vstrnc < maxtrc OR vspwrd.vslev < maxlev THEN
```

```
            BEGIN
            IF vspwrd.vstrnc < maxtrc THEN
               *sfhead* _ *sfhead*, " T=", STRING(vspwrd.vstrnc), ",
            ELSE *sfhead* _ *sfhead*, " T=ALL,";
            IF vspwrd.vslev < maxlev THEN
               *sfhead* _ *sfhead*, " L=", STRING(vspwrd.vslev), ",
            ELSE *sfhead* _ *sfhead*, " L=ALL,";
            END;
         %get file name%
            filnam(da.dacsp.stfile, $filstr);
         %blank fill to position for file name%
            FOR count _ 1 UP UNTIL > MAX (1, sfhpgn - filstr.L - sfhead.L)
            DO
               *sfhead* _ *sfhead*, SP;
         %put file name in header%
            *sfhead* _ *sfhead*, *filstr*;
         %2 blanks to position for page number%
            *sfhead* _ *sfhead*, "  ";
         RETURN;
         END.

%.....insert sequential.....%

   (inseq) PROCEDURE   % does the Copy Sequential command %                6A
      (stid,      % statement after which to insert %
      rlevcnt,    % level relative to stid %
      itilnam,    % string containing the name of the input file %
      filtyp);    % type of input %

      % This is the main control routine for insert sequential.  The
      input file is opened and closed here. %
      %------------------%
      LOCAL count, levblk, lstblk, curblk, curdpth, levchg, char, pos,
      offset, jfn, t1, t2, t3, t4, t5, t6, term, firststid;
      LOCAL TEXT POINTER start, end;
      REF ifilnam;

      dismes(1, $"Copy Sequential in Progress");
      firststid_0;
      % open the input file %
         IF NOT jfn _ lgetjfn (0, &ifilnam, $txtext, gtjoisf, $lit)
         THEN
            ABORT (ofilerr, $lit);
         IF NOT sysopen (jfn, read, chrtyp, $lit) THEN
            BEGIN
            reljfn (jfn);
            ABORT (ofilerr, $lit);
            END;
      IF filtyp = assfil THEN GOTO inseqaws;
      levblk _ lstblk _ curdpth _ 0;
      offset _ IF filtyp = n40fil THEN n40fst ELSE tenfst;
      *lit* _ NULL;
      isread (jfn, $sar, CR);
      pos _ empty + 1;
      WHILE *sar*[pos] = LF DO BUMP pos;
      IF filtyp # macfil THEN
```

```
      CASE *sar*[pos]  OF
         =ascmbk:
            levblk _ lstblk _ *sar*[pos _ pos + 1] - offset;
         =SP, =TAB:
            BEGIN
            levblk _ 1;
            LOOP
               CASE *sar*[pos _ pos + 1] OF
                  =SP, =TAB: BUMP levblk;
                  ENDCASE EXIT LOOP;
            lstblk _ levblk;
            pos _ pos - 1;
            END;
         ENDCASE pos _ pos - 1
   ELSE pos _ pos - 1;
   curdpth _ 0;
   LOOP
      BEGIN
      %loop end condition changed from > to = to compensate for old
      compiler bug%
      UNTIL (pos := pos + 1) = sar.L DO
         CASE char _ *sar*[pos] OF
            =ascmbk:
               BEGIN
               count _ *sar*[pos _ pos + 1] - offset;
               UNTIL (count := count - 1) = 0 DO
               *lit* _ *lit*, SP;
               END;
            = LF, = 0: NULL;
            = ENDCHR: IF lit.L NOT= empty THEN GOTO insql1 ELSE GOTO
            insql2;
            = EOL, =CR: %end of statement%
               BEGIN
               (insql1):                                        6A23C1D2
                  FIND SF(*lit*) ^start SE(*lit*) ^end;
                  stid _ <STRMNP, cinssta> (stid, rlevcnt, $start,
                  $end);
                  IF firststid=0 THEN firststid_stid;
               (insql2): *lit* _ NULL;                          6A23C1D3
               IF NOT isread(jfn, $sar, CR) THEN EXIT LOOP 2;
               pos _ empty + 1;
               WHILE *sar*[pos] = LF DO BUMP pos;
               IF filtyp # macfil THEN
                  CASE *sar*[pos] OF
                     %determine level of next statement%
                     =ascmbk:
                        curblk _ *sar*[pos _ pos + 1] - offset;
                     =SP, =TAB:
                        BEGIN
                        count _ 1;
                        LOOP
                           CASE *sar*[pos _ pos + 1] OF
                              =SP, =TAB:
                                 BEGIN
                                 IF filtyp = macfil AND count >=
                                 maxmclv THEN EXIT LOOP;
```

```
                                BUMP count;
                                END;
                             ENDCASE EXIT LOOP;
                          pos _ pos - 1;
                          curblk _ count;
                          END;
                       ENDCASE
                          BEGIN
                          curblk _ 0;
                          pos _ pos - 1;
                          END
                 ELSE
                    BEGIN
                    curblk _ 0;
                    pos _ pos - 1;
                    END;
                 IF curblk THEN
                    BEGIN
                    IF NOT levblk THEN levblk _ curblk;
                    IF filtyp # macfil THEN
                       CASE lstblk OF
                          =curblk: %same level%
                             rlevcnt _ levsuc;
                          < curblk: %levdown a level%
                             IF (curblk - lstblk) >= levblk
                             OR curblk/levblk > curdpth THEN
                                BEGIN
                                rlevcnt _ levdown;
                                BUMP curdpth;
                                END
                             ELSE rlevcnt _ levsuc;
                          ENDCASE %up a level%
                             BEGIN
                             levchg _ (lstblk - curblk) / levblk;
                             IF levchg < 1 AND curdpth > 2 THEN
                                levchg _ (curdpth - 1) -
                                curblk/levblk;
                             IF levchg > curdpth THEN levchg _
                             curdpth;
                             curdpth _ curdpth - levchg;
                             rlevcnt _ levsuc;
                             WHILE (levchg _ levchg - 1) >= 0 DO
                                rlevcnt _ rlevcnt + 1;
                             END;
                    END
                 ELSE
                    BEGIN
                    rlevcnt _ levsuc;
                    IF lstblk THEN
                       WHILE (curdpth := curdpth - 1) > 0 DO
                          rlevcnt _ rlevcnt + 1;
                    curdpth _ 0;
                    END;
                 lstblk _ curblk;
                 END;
              ENDCASE *lit* _ *lit*, char;
```

```
      IF NOT isread(jfn, $sar, CR) THEN EXIT LOOP;
      pos _ empty;
      END;
   IF lit.L NOT= empty THEN
      BEGIN
      FIND SF(*lit*) ^start SE(*lit*) ^end;
      stid _ <STRMNP, cinssta> (stid, rlevcnt, $start, $end);
      END;
   %close file%
      (insegcl):                                                    6A25A
      IF NOT sysclose (jfn, $lit) THEN
         BEGIN
         reljfn (jfn);
         err($lit);
         END;
   dismes(0);
   IF firststid=0 THEN firststid_stid;
   RETURN(firststid);

   (insegaws):                                                      6A30

   % the following code is concerned with insert sequential
   assembler file with structure.  the first statment in the
   assembler file is inserted without modification at the level
   specified by the user.  succeeding statements are placed one
   level levdown for each tab at the front of the statement and up
   the appropriate number of levels if a succeeding statement has
   less tabs than the current statement.  ASCIZ, ASCII, and SIXBIT
   statements are handled as follows:  the first non-blank
   character following the delimiter character for the ASCIZ, ASCII,
   and SIXBIT pseudo-ops serves as a delimiter.  all characters
   between this delimiter and its next occurrence are inserted
   literally in the statement, i.e., EOL or CR do not serve to
   delimit the statement, and no level adjustments are made. %

   % insert the first statement %
   *lit* _ NULL;
   IF NOT isread(jfn, $sar, CR) THEN GOTO iswsend;
   pos _ empty + 1;
   WHILE *sar*[pos] = LF  DO BUMP pos;
   curdpth _ 0;
   LOOP
      CASE char _ *sar*[pos := pos + 1]  OF
         = LF: NULL;
         = ENDCHR:  % the only way we get here is end of file %
            BEGIN
            IF lit.L NOT= empty THEN
               BEGIN
               FIND SF(*lit*) ^start SE(*lit*) ^end;
               cinssta(stid, rlevcnt, $start, $end);
               END;
            GOTO insegcl;  % go close the file %
            END;
         = EOL, =CR:
            BEGIN
            FIND SF(*lit*) ^start SE(*lit*) ^end;
```

```
            stid _ cinssta(stid, rlevcnt, $start, $end);
            EXIT LOOP;
            END;
         ENDCASE *lit* _ *lit*, char;

   LOOP   % this is the main loop for the rest of the statements %
      BEGIN

      % initialization %
      *lit* _ NULL;
      rlevcnt _ levsuc;

      % read in a statement from the assembler file %
      IF NOT isread(jfn, $sar, CR) THEN EXIT LOOP;
         % i.e. GOTO iswsend %
      pos _ empty + 1;

      % bypass LF %
      WHILE *sar*[pos] = LF DO BUMP pos;

      % find out level of statement relative to previous statement %
      count _ 0;
      WHILE *sar*[pos] = TAB DO
         BEGIN
         BUMP count;
         BUMP pos;
         END;
      CASE count OF
         > curdpth:
            BEGIN
            rlevcnt _ levdown;
            count _ count - curdpth;
            curdpth _ curdpth + 1;
            WHILE (count _ count - 1) > 0 DO *lit* _ *lit*, TAB;
            END;
         < curdpth:
            BEGIN
            t1 _ curdpth - count;
            WHILE (t1 := t1 - 1 ) > 0 DO rlevcnt _ rlevcnt +1;
            curdpth _ count;
            END;
         ENDCASE;

      % we are now past any leading tabs and will gobble up the rest
      of the statement one character at a time %
      LOOP
         CASE char _ *sar*[pos := pos + 1] OF
            = LF: NULL;
            = ENDCHR:  % the only way we get here is end of file %
               BEGIN
               IF lit.L NOT= empty THEN
                  BEGIN
                  FIND SF(*lit*) ^start SE(*lit*) ^end;
                  cinssta(stid, rlevcnt, $start, $end);
                  END;
               GOTO insegcl;  % go close the file %
```

```
            END;
         = EOL, = CR:
            BEGIN
            FIND SF(*lit*) ^start SE(*lit*) ^end;
            stid _ cinssta(stid, rlevcnt, $start, $end);
            EXIT LOOP;
            END;
         = 'A, ='a:    % check for asciz or ascii %
            BEGIN
            *lit* _ *lit*, char;
            IF pos + 4 <= sar.L THEN
               BEGIN
               t1 _ *sar*[pos + 0];
               t2 _ *sar*[pos + 1];
               t3 _ *sar*[pos + 2];
               t4 _ *sar*[pos + 3];
               t5 _ *sar*[pos + 4];
               IF ( (t1 = 'S) OR (t1 = 's) ) AND
                  ( (t2 = 'C) OR (t2 = 'c) ) AND
                  ( (t3 = 'I) OR (t3 = 'i) ) AND
                  ( (t4 = 'Z) OR (t4 = 'z) OR
                    (t4 = 'I) OR (t4 = 'i) ) AND
                  ( (t5 NOT IN ['A, 'Z]) AND
                    (t5 NOT IN ['a, 'z]) AND
                    (t5 # '$) AND (t5 # '%) AND
                    (t5 # '.) AND (t5 # LF) AND
                    (t5 # EOL) AND (t5 # CR) )  THEN
                  BEGIN
                  *lit* _ *lit*, t1, t2, t3, t4, t5;
                  pos _ pos + 4;
                  IF (t5 # SP) AND (t5 # TAB) THEN term _ t5
                  ELSE LOOP
                     IF (pos _ pos + 1) <= sar.L THEN
                        IF ((term _ *sar*[pos]) = SP) OR
                           (term = TAB) THEN *lit* _ *lit*, term
                        ELSE
                           BEGIN
                           *lit* _ *lit*, term;
                           EXIT LOOP;
                           END
                     ELSE
                        BEGIN
                        IF NOT isread(jfn, $sar, CR) THEN
                           GOTO iswsend;
                        pos _ empty;
                        END;

                  % We have now gobbled up asciz or ascii and the
                  first delimiter and are gobbling succeeding
                  characters until we find the second occurrence
                  of the delimiter %
                  LOOP
                     CASE char _ *sar*[pos _ pos + 1] OF
                        = LF: NULL;

                           % the only way we get here is end of
```

```
                        file%
                        = ENDCHR:
                           BEGIN
                           IF lit.L NOT= empty THEN
                              BEGIN
                              FIND SF(*lit*) ^start SE(*lit*)
                              ^end;
                              cinssta(stid, rlevcnt, $start,
                              $end);
                              END;
                           GOTO inseqcl;  % go close the file %
                           END;
                        = EOL, = CR:
                           BEGIN
                           *lit* _ *lit*, EOL;
                           IF NOT isread(jfn, $sar, CR) THEN
                              GOTO iswsend;
                           pos _ empty;
                           END;
                        = term:
                           BEGIN
                           *lit* _ *lit*, char;
                           pos _ pos + 1;
                           EXIT LOOP;
                           END;
                        ENDCASE *lit* _ *lit*, char;
                     END;
                  END;
               END;
            = 'S, ='s:   % check for sixbit %
               BEGIN
               *lit* _ *lit*, char;
               IF pos + 5 <= sar.L THEN
                  BEGIN
                  t1 _ *sar*[pos + 0];
                  t2 _ *sar*[pos + 1];
                  t3 _ *sar*[pos + 2];
                  t4 _ *sar*[pos + 3];
                  t5 _ *sar*[pos + 4];
                  t6 _ *sar*[pos + 5];
                  IF ( (t1 = 'I) OR (t1 = 'i) ) AND
                     ( (t2 = 'X) OR (t2 = 'x) ) AND
                     ( (t3 = 'B) OR (t3 = 'b) ) AND
                     ( (t4 = 'I) OR (t4 = 'i) ) AND
                     ( (t5 = 'T) OR (t4 = 't) ) AND
                     ( (t6 NOT IN ['A, 'Z]) AND
                       (t6 NOT IN ['a, 'z]) AND
                       (t6 # '$) AND (t6 # '%) AND
                       (t6 # '.) AND (t6 # LF) AND
                       (t6 # EOL) AND (t6 # CR) ) THEN
                     BEGIN
                     *lit* _ *lit*, t1, t2, t3, t4, t5, t6;
                     pos _ pos + 5;
                     IF (t6 # SP) AND (t6 # TAB) THEN term _ t6
                     ELSE LOOP
                        IF (pos _ pos + 1) <= sar.L THEN
```

```
                        IF ((term _ *sar*[pos]) = SP) OR
                           (term = TAB) THEN *lit* _ *lit*, term
                        ELSE
                           BEGIN
                           *lit* _ *lit*, term;
                           EXIT LOOP;
                           END
                     ELSE
                        BEGIN
                        IF NOT isread(jfn, $sar, CR) THEN
                           GOTO iswsend;
                        pos _ empty;
                        END;

                  % we have now gobbled up sixbit and the first
                  delimiter and are gobbling succeeding
                  characters until we find the second occurrence
                  of the delimiter %
                  LOOP
                     CASE char _ *sar*[pos _ pos + 1] OF
                        = LF: NULL;

                        % the only way we get here is end of
                        file%
                        = ENDCHR:
                           BEGIN
                           IF lit.L NOT= empty THEN
                              BEGIN
                              FIND SF(*lit*) ^start SE(*lit*)
                              ^end;
                              cinssta(stid, rlevcnt, $start,
                              $end);
                              END;
                           GOTO inseqcl;  % go close the file %
                           END;
                        = EOL, = CR:
                           BEGIN
                           *lit* _ *lit*, EOL;
                           IF NOT isread(jfn, $sar, CR) THEN
                              GOTO iswsend;
                           pos _ empty;
                           END;
                        = term:
                           BEGIN
                           *lit* _ *lit*, char;
                           pos _ pos + 1;
                           EXIT LOOP;
                           END;
                        ENDCASE *lit* _ *lit*, char;
                  END;
               END;
            END;
         ENDCASE *lit* _ *lit*, char;
   END;

(iswsend):   % cleanup at end of file %                    6A44
```

```
    t1 _ sar.L;
    pos _ empty;
    WHILE (t1 := t1 - 1) > 0 DO *lit* _ *lit*, *sar*[pos _ pos + 1];
    IF lit.L NOT= empty THEN
       BEGIN
       FIND SF(*lit*) ^start SF(*lit*) ^end;
       cinssta(stid, rlevcnt, $start, $end);
       END;
    GOTO insegcl;    % close the file and return %

    END.
```

```
(isread)                                                              6B
    %This procedure is used during insert sequential to read another
    character from the input file.  It expects the input jfn, the
    address of a string, and a terminating character, as arguments,
    in that order.  It puts the characters read into the string; it
    returns true until it has an empty input buffer and encounters an
    EOF.%
    %------------------%
    PROCEDURE(jfn, inbuff, lstchr);
    LOCAL error, maxcnt, char, count, bytptr;
    REF inbuff;
    bytptr _ chbnty + &inbuff;
    maxcnt _ inbuff.M;
    count _ empty;
    LOOP
       BEGIN
       !bin( jfn );
       IF NOT (char _ R2 .A 377B) % null byte % THEN
          BEGIN
          !gtsts( jfn );
          error _ R2;
          IF error .A seger # 0 THEN% Input error %
             BEGIN
             IF NOT sysclose (jfn, $lit) THEN
                BEGIN
                reljfn (jfn);
                err ($lit);
                END
             ELSE err ($"Input File Error");
             END;
          IF error .A segeof # 0 THEN % EOF read-- check if buffer
          empty %
             BEGIN
             IF (inbuff.L _ count) > empty THEN RETURN(TRUE)
             ELSE RETURN( FALSE );
             END;
          END;
       ^bytptr _ char;
       IF ((count _ count + 1) >= maxcnt) OR (char = lstchr) OR (char
       = EOL AND lstchr = CR) THEN
          BEGIN
          inbuff.L _ count;
          RETURN(TRUE);
          END;
```

```
      END;
   END.

(inseqn) PROCEDURE   %does the input sequential command (new way):
double CR means end of st.%                                          6C
   (stid,      % statement after which to insert                     %
    rlevcnt,   % level relative to stid                              %
    ifilnam,   % string containing name of input file                %
    just);     % were multiple spaces added to right justify?
   ------------------------------------------------------------------%
   % declarations %
      LOCAL jfn, place, dummy, indent1, indent2, level, adj, done ;
      LOCAL TEXT POINTER stop, end, t1, t2, z1, z2 ;
      LOCAL STRING line1[999], line2[999], st[2000], string[20] ;
      REF ifilnam ;
      level _ done _ 0;
   dismes (1, $"Insert Sequential in Progress ");
   % handle different types %
   % open input file %
      IF NOT jfn _ lgetjfn (0, &ifilnam, $txtext, gtjoisf, $lit)
         THEN ABORT (ofilerr, $lit) ;
      IF NOT sysopen (jfn, read, chrtyp, $lit) THEN
         BEGIN
         reljfn (jfn) ;
         ABORT (ofilerr, $lit) ;
         END;
   % insert dummy statement %
      *string* _ "dummy";
      FIND SF(*string*) ^t1 SE(*string*) ^t2;
      dummy _ place _ cinssta (stid, rlevcnt, $t1, $t2) ;
   % for all data in input file %
      LOOP %each cycle a statement%
         BEGIN
         adj _ indent1 _ indent2 _ 0 ;
         % get first line %
            DO %skip blank lines%
               BEGIN
               indent1 _ inseql (jfn, $line1, just) ;
               IF indent1 = -1 THEN
                  BEGIN
                  done _ 1;
                  EXIT LOOP;
                  END;
               END
            UNTIL line1.L>0 ;
         % get second line %
            indent2 _ inseql (jfn, $line2, just) ;
            IF indent2 = -1 THEN
               done _ 1;
         % add both lines to *st* %
            *st* _ *line1*, *line2* ;
         % figure out level %
            IF (line2.L AND indent2) > 0 THEN indent1 _ indent2;  %
            We go by the second line in all cases except where there
            is no second line or it is not indented.  Then we go by
            the first line. %
```

```
            CASE indent1 OF
               > levind[level]:
                  IF level = 11 THEN
                     adj _ levsuc
                  ELSE
                     BEGIN
                     BUMP level;
                     levind[level] _ indent1 ;
                     IF (place := getsub(place)) = place
                        THEN adj _ levdown
                        ELSE
                           BEGIN
                           place _ getail(place);
                           adj _ levsuc;
                           END;
                     END;
               < levind[level]:
                  BEGIN
                  BUMP DOWN level ;
                  place _ getup (place) ;
                  REPEAT CASE;
                  END;
               ENDCASE  % =levind[level] %
                  adj _ levsuc ;
         % add lines to *st* until double EOL %
            IF line2.L > 0 THEN
               LOOP
                  BEGIN
                  IF inseql(jfn,$line1,just) = -1 THEN
                     BEGIN
                     done _ 1;
                     EXIT LOOP;
                     END;
                  IF line1.L=0 THEN EXIT LOOP;
                  IF (st.L + line1.L) > st.M THEN
                     BEGIN  %start new statement%
                     FIND SF(*st*) ^z1 SE(*st*) ^z2;
                     place _ cinssta (place, adj, $z1, $z2) ;
                     adj _ levsuc ;
                     st.L _ 0 ;
                     END;
                  *st* _ *st*, *line1* ;
                  END;
         % insert the statement %
            FIND SE(*st*) ^end $NP ^stop > ;
            ST stop end _ NULL ;
            FIND SF(*st*) ^z1 SE(*st*) ^z2;
            place _ cinssta (place, adj, $z1, $z2) ;
            IF done THEN EXIT LOOP;
         END;
   % close the input file %
      IF NOT sysclose (jfn, $lit) THEN
         BEGIN
         reljfn (jfn) ;
         dismes (2, $lit) ;
         ABORT( statesig, 0);
```

```
        END;
    % delete dummy statement %
      IF (place _ getsub(dummy)) # dummy THEN
        cmovgro (dummy, levsuc, gethed(place) , getail(place),
        FALSE, 0) ;
      place _ getsuc(dummy);
      cdelsta (dummy, FALSE, 0) ;
    dismes(0) ;
    RETURN(place);
    END.
(inseql) PROCEDURE (jfn, line, justified) ; %gets a line for inseqn%
                                                                    6D
    % declarations %
      LOCAL indent, short ;
      LOCAL TEXT POINTER pt1, pt2 ;
      REF line ;
    % get line %
      line.L _ 0 ;
      IF NOT isread (jfn, &line, CR) THEN RETURN (-1); %couldn't get
      line%
      short _ (IF line.L < 50 THEN TRUE ELSE FALSE) ;
    % if within 8 chars of RM, and if fully justified, remove extra
    spaces if less than or equal to 5 in a row %
      IF justified AND NOT short THEN
        BEGIN
        FIND SF(*line*) $NP ;
        LOOP
          BEGIN
          IF NOT FIND [2SP] THEN EXIT LOOP;
          IF FIND ^pt1 _pt1 $3SP ^pt2 PT THEN
            ST pt1 pt2 _ NULL
          ELSE FIND $SP;
          END;
        END;
    % count and delete leading spaces %
      CCPOS SF(*line*) ;
      indent _ 0;
      CASE READC OF
        =SP:
          BEGIN
          indent _ indent + 1 ;
          REPEAT CASE ;
          END;
        =TAB:
          BEGIN
          indent _ indent + 8 ;
          REPEAT CASE ;
          END;
        =NP:  REPEAT CASE;
        ENDCASE FIND ^pt1 _pt1;
    % delete trailing spaces %
      CCPOS SE(*line*) ;
      CASE READC OF
        =CR:
          IF short THEN
            BEGIN
```

```
                FIND ^pt2 >;
                *line* _ pt1 pt2, EOL ;
                IF line.L=1 THEN line.L _ 0 ;
                END
             ELSE REPEAT CASE ;
          =NP:  REPEAT CASE;
          ENDCASE
             IF line.L < line.M THEN
                BEGIN
                FIND ^pt2 _pt2 >;
                *line* _ pt1 pt2, SP ;
                END;
   RETURN (indent);
   END.
(hiop) PROCEDURE (acs); % NLS Host Input/Output Processor %          6E
   % Declarations %
      LOCAL
         direction, % file entering/leaving system %
         parmchr,   % single char of parms %
         sysmsg,  %address of signal message arg%
         i,         % working index variable %
         injfn,     % source file jfn %
         outjfn,    % result file jfn %
         bp1, bp2;  % byte pointers %
      LOCAL TEXT POINTER
         tp1, tp2, tp3, tp4, z1, z2; % working pointers %
      LOCAL STRING
         parms [40],     % parameters %
         tail [40],      % tail of parms %
         infile [40],    % source filename %
         outfile [40],   % result filename %
         fieldname [40], % name of field %
         fieldvalue [40],% value of field %
         viewspecs [40], % viewspecs string %
         addr [40],      % statement addr %
         convtype [40],  % conversion algorithm %
         errstr [40];    % error string %
      REF acs; % AC's upon entry to NLS %
   % Save direction and JFN %
      INVOKE (sigset);
      R1 _ acs [7]; % get xwd direction, input jfn %
      ! HLRE 2,1; % isolate direction, propagate sign %
      ! HRRZS 1; % clean input jfn %
      injfn _ R1; % save input jfn %
      direction _ R2; %save direction %
   % Build parameter string %
      bp1 _ chbptr(empty) + $parms;
      bp2 _ chbptr(empty) + &acs + 7;
      R1 _ bp1; % load destination AC %
      R2 _ bp2; % load source AC %
      R3 _ 0; % terminate on NUL %
      ! JSYS sout; % copy the ASCIZ string %
      bp2 _ R1; % save dest-end pointer %
      parms.L _ slngth (bp1, bp2); % set parms length %
   % Init defaults %
      *viewspecs* _ ""; % null viewspecs %
```

```
      *convtype* _ "S"; % convert to/from sequential file %
      *addr* _ ".1"; % point to top of file %
   % Extract fields from parameter string %
      CCPOS *parms*; % init text pointer %
      *tail* _ *parms*; % init tail of parms %
      IF parms.L # 0 THEN % skip parse if null parms %
         LOOP
            BEGIN % here for each field of parms %
            IF NOT FIND ^tp1 $LD ^tp2 ": ^tp3 ((L",] ^tp4 _tp4) /
            (SE(tp3) ^tp4 >)) THEN hiortn (cat, $"What is ", $tail,
            $"?"); % extract next field and its value %
            *fieldname* _ + tp1 tp2; % save name of field %
            *fieldvalue* _ tp3 tp4; % and its value %
            IF fieldname.L # 1 THEN GOTO badfield; % all field types
            are one char %
            CASE *fieldname* [1] OF % save field value appropriately
            %
               ="T: *convtype* _ + SF(*fieldvalue*)
               SE(*fieldvalue*); % conversion type %
               ="V: *viewspecs* _ *fieldvalue*; % viewspecs %
               ="N: *addr* _ *fieldvalue*; % statement %
               ENDCASE (badfield): hiortn (cat, $fieldname, $"-field
               undefined", $""); % undefined field type %
            *tail* _ tp4 SF(tp4); % parms all processed? %
            IF tail.L = 0 THEN EXIT LOOP; % quit if so %
            END;
   % Load source file %
      DROP (sigset);
      INVOKE (sigload);
      jfnstr (injfn, $infile); % fetch filename %
      tda.dacsp _ orgstid;  tda.daccnt _ 1;
      tda.dacsp.stfile _ cloafil ($infile); % load source file %
   % Prepare output file %
      DROP (sigload);
      INVOKE (sigout);
      IF NOT outjfn _ sgtjfn (6B11, $"HIOPWORK.TXT", $errstr) THEN
      hiortn (cat, $"Problems with output file: ", $errstr, $""); %
      abort if problems with output file %
      jfnstr (outjfn, $outfile); % fetch output filename %
   % Verify and instate viewspecs %
      DROP (sigout);
      INVOKE (sigvs);
      feedlt (&tda, $viewspecs); % build viewspecs %
   % Verify and position to statement addr %
      DROP (sigvs);
      INVOKE (sigaddr);
      b1 _ tda.dacsp; % get current... %
      b1 [1] _ tda.daccnt; % ...command marker %
      FIND SF(*addr*) ^z1 SE(*addr*) ^z2;
      caddexp ($z1, $z2, &tda, $b1); % interpret TNLS addr %
      tda.dacsp _ b1;  tda.daccnt _ b1[1];
   % Invoke the conversion %
      DROP (sigaddr);
      INVOKE (sigconv);
      IF direction # leaving THEN hiortn (cat, $"Input not yet
      supported", $"", $""); % file input not yet implemented %
```

```
CASE *convtype* [1] OF
   ='A: outasm ($outfile, &tda, FALSE);
   ='P: outproc ($outfile, &tda, opprdv, 0);
   ='Q: outqp ($outfile, &tda);
   ='S: outseq ($outfile, &tda, FALSE);
   ENDCASE hiortn (cat, $convtype, $"-conversion type
   undefined", $"");
DROP (sigconv);
hiortn (outjfn, $convtype, $"-conversion", $""); % successful
completion %

(sigset) CATCHPHRASE(:sysmsg);                                  6E10H
   BEGIN
   DISABLE (sigset);
   hiortn (cat, $"Conversion setup error: ", sysmsg, $""); %
   catch signals %
   CONTINUE;
   END;

(sigload) CATCHPHRASE(:sysmsg);                                 6E10J
   BEGIN
   DISABLE (sigload);
   hiortn (cat, $"Can't load file: ", sysmsg, $""); % catch
   signals %
   CONTINUE;
   END;

(sigout) CATCHPHRASE(:sysmsg);                                  6E10L
   BEGIN
   DISABLE (sigout);
   hiortn (cat, $"Can't get output file: ", sysmsg, $""); %
   catch signals %
   CONTINUE;
   END;

(sigvs) CATCHPHRASE(:sysmsg);                                   6E10N
   BEGIN
   DISABLE (sigvs);
   hiortn (cat, $viewspecs, $" are invalid viewspecs: ",
   sysmsg); % catch signals %
   CONTINUE;
   END;

(sigaddr) CATCHPHRASE(:sysmsg);                                 6E10P
   BEGIN
   DISABLE (sigaddr);
   hiortn (cat, $addr, $" is an invalid addr: ", sysmsg); %
   catch signals %
   CONTINUE;
   END;

(sigconv) CATCHPHRASE(:sysmsg);                                 6E10R
   BEGIN
   DISABLE (sigconv);
   hiortn (cat, $convtype, $"-conversion error: ", sysmsg); %
   catch signals %
```

```
            CONTINUE;
            END;

         END.

%.....output device and output compiler.....%

%.....processor dispatch.....%

   (processor) PROCEDURE   % start and run a processor %                    3A
      (prcnam,  % string containing the name of the processor %
      tp,       % text pointer to first thing to feed the processor %
      da,       % display area descriptor of the input to processor %
      type,     % type of the processor %
      outdsg,   % output designator:  either the address of a buffer or
      the jfn of an output file %
      prcjfn);  %JFN of processor if no processor name pased or 0%

         % ****** The Catalog System has its own version of PROCESSOR
         so please tell Walt if you make any changes ****** %


      LOCAL stid, size, error, savpreg, savwareg, vspec, sdb, dirnam,
      subsbn;
      LOCAL TEXT POINTER fchartp, tptr;
      LOCAL STRING ssysnam[40];
      REF prcnam, tp, da, sdb, dirnam;

      % set up to get first character for the processor %
         vspec _ da.davspec;
         &prseqwk _ 0;
         IF type = cmptyp THEN vspec.vsstnf _ TRUE;
            % turn statement numbers on for MPL %
         &prseqwk  _ openseq (tp, seqend(tp, vspec, da.davspc2), vspec,
         da.davspc2, da.dausgcod, IF tp.stastr THEN 0 ELSE
         da.dacacode);
         INVOKE(sclose);
         IF (stid _ seqgen(&prseqwk)) = endfil THEN
            BEGIN
            DROP(sclose);
            closeseq (&prseqwk:=0);
            ABORT (prcerr, $"No processor input");
            END;
         IF tp.stastr THEN FIND (tp ^fchartp)
         ELSE FIND (SF(stid) ^fchartp);
         %-NSW%                                                          8A11I
            &dirnam _ $subdir;
         %-NSW%                                                          8A11J
         %+NSW%                                                          8A11K
            &dirnam _ 0;
         %+NSW%                                                          8A11L
         IF NOT prcjfn AND NOT (prcjfn _ lgetjfn(&dirnam, &prcnam,
         $savext, gtjprf, $lit))
            THEN ABORT (prcerr, $lit);
      % get a string containing name of the processor %
         IF type = upgtyp THEN *ssysnam* _ "NLSL10"
```

```
      ELSE
         BEGIN
         jfntostr (prcjfn, $ssysnam, 11B9);
         IF NOT FIND SF(*ssysnam*) "SUBSYS" ^tptr THEN
            *ssysnam* _ "PRVPRC"
         ELSE *ssysnam* _ tptr SE(*ssysnam*);
         END;
   %map out high pages%
      hmapout();
   %get and save current subsystem name%
      !getnm();
      subsbn _ R1;
   % set the subsystem name %
      R1 _ getsbn ($ssysnam);
      !JSYS setnm;
   % "get" the processor %
      R1.LH _ 4B5;
      R1.RH _ prcjfn;
      !JSYS get;
   IF [prcadr + 1] THEN chntab[9] _ [prcadr + 1] .V 1B6;
   savpreg _ p; savwareg _ wa;  % save wa and p registers since the
   compilers and the Output Processor will clobber them %
   INVOKE (prcrstr, procrestore);
   error _ 0;
   CASE type OF
      = upgtyp:    % compile a user program %
         BEGIN
         IF NOT cppflag THEN dismes (1, $"Compiling User Program");
         swork _ fchartp; swork[1] _ fchartp[1]; fechc1(1, $swork);
         %swork[2] contains byte count+1, swork[4] has byte pointer%
         error _ [[prcadr]]
            (-1, $prgetst, outdsg, , $lev1lc, swork[4], swork[2]-1
            :size);
         %error _ sprocessor
            (-1, $zprgetst+3, outdsg, , $lev1lc, swork[4],
            swork[2]-1 :size);%
         END;
      = cmptyp:    % compile a program into an output file %
         BEGIN
         dismes (1, $"Compiling");
         swork _ fchartp; swork[1] _ fchartp[1]; fechc1(1, $swork);
         %swork[2] contains byte count+1, swork[4] has byte pointer%
         error _ [[prcadr]]
            (outdsg, $prgetst, prseqwk.swsvw, , $lev1lc, swork[4],
            swork[2]-1);
         %error _ sprocessor
            (outdsg, $zprgetst+3 , prseqwk.swsvw, , $lev1lc,
            swork[4], swork[2]-1);%
         END;
      = odtyp:     % output device type %
         BEGIN
         dismes (1, IF exp
            THEN $"Experimental Output Processor"
            ELSE $"Processing Output");
         IF NOT stid.stastr THEN
            BEGIN
```

```
            IF NOT lodprop( stid, txttyp : &sdb) THEN
               err($"No text block associated with node");
            opbp _ opcbp _ &sdb + sdbhdl + 4407B8;
            opcmax _ sdb.schars + 1;
            END
         ELSE
            BEGIN
            opbp _ opcbp _ chbmty + stid.stpsid;;
            opcmax _ [stid.stpsid].L + 1;
            END;
         opccpos _ 1;
         opnewst _ TRUE;
         error _ [[prcadr]] ($oprwrk, $lev1lc, opbp, opcmax);
         END;
      ENDCASE   % some other type of processor %
         BEGIN
         dismes (1, $"Processor in progress");
         swork _ fchartp; swork[1] _ fchartp[1]; fechc1(1, $swork);
         %swork[2] contains byte count+1, swork[4] has byte pointer%
         error _ [[prcadr]] (outdsg, $prgetst, , , $lev1lc,
         swork[4], swork[2]-1);
         END;
   (procrestore):                                                  8A22
   DROP (prcrstr);
   % restore wa and p registers %
      P _ savpreg; wa _ savwareg;
   %restore stack overflow pseudo-interrupt%
      chntab[9] _ $sysovr .V 1B6;
   %get nls pages back%
      hmapin(); % also releases prcjfn!!! %
   %restore subsystem name to NLS%
      R1 _ subsbn;
      !JSYS setnm;
   %close the sequence%
      DROP(sclose);
      IF &prseqwk THEN closeseq (&prseqwk:=0);
   %reset entry vector back to NLS%
      setvec();
   IF inptrf THEN %^O pseudo-interrupt%
      BEGIN
      rstcntlo(); % tell FE to resume output %
      dismes( 1, $"User Terminated Process");
      END
   ELSE
      %take message away%
         dismes (0);
   RETURN (error, size);

   (prcrstr) CATCHPHRASE;                                          8A34
      BEGIN
      DISABLE (prcrstr);
      CASE SIGNALTYPE OF
         =aborttype:
            BEGIN
            TERMINATE; % will get to PROCRESTORE label %
            END;
```

```
      ENDCASE CONTINUE;
      END;

   (sclose) CATCHPHRASE;                                          8A36
      BEGIN
      DISABLE (sclose); % In case closeseq generates a signal %
      %close the sequence%
         IF &prseqwk THEN closeseq (&prseqwk:=0);
      IF inptrf THEN rstcntlo();
      CONTINUE;
      END;

   END.

(mapped) RECORD % pmap handle and page status. %                   8B
   fhandC26], % handle from handle. %
   hndaccC10]; % access information. %
(hmapout)PROC;                                                    8C
   mapout( $bfree/1000B, $efree/1000B );
   RETURN;
   END.

(mapout)PROC( startpage, endpage );                               8D
   LOCAL count, fbase, handle;
   %Keep track of pages mapped out%
      FOR count _ (fbase _ startpage) UP UNTIL >= endpage DO
         BEGIN
         R1.LH _ 485;
         R1.RH _ count;
         !JSYS rpacs;
         IF SKIP !TLNN R2, 10000B THEN %page exists%
            BEGIN
            %get handle for page%
               R3 _ R2;  %save it.  not set correctly by rmap jsys%
               !JSYS rmap;
               R2 _ R3;  %restore R2%
               IF R1.LH = 485 THEN pmfmap(TRUE);
                  %the pmf stuff doesn't work with tops20 so we have
                  to hndle it%
               handle _ R1;
               handle.hndacc _ R2.hndacc;
            %remove page%
               R2.LH _ 485;
               R2.RH _ count;
               R1 _ -1;
               R3 _ 0;
               !JSYS pmap;
            END
         ELSE  %page does not exist%
            handle _ -1;
         freemapCcount-fbase] _ handle;
         END;
   RETURN;
   END.

(pmtmap)PROCEDURE(removf);                                        8E
```

```
   %map fork page in R1 to "UPMF" file and return new handle in R1.
   R2 contains access, returns it in tact.%
   LOCAL STRING filnms[100], errstr[200];
   !PUSH S,R1; !PUSH S,R2;
   IF NOT pmfjfn THEN %must get the file%
      BEGIN
      *filnms* _ "<", *userstr*, ">", "$UPMF$.PGS;P707000;T", 0;
      IF (pmfjfn _ sgtjfn(0, $filnms, $errstr)) AND NOT SKIP
      !openf(pmfjfn, 302000B) THEN reljfn(pmfjfn:=0);
      IF NOT pmfjfn THEN err($"Can't open PMF file");
      END;
   !ffffp(pmfjfn); %get a free page%
   R2 _ R1; %pmap destination%
   !POP S,R3; %access%
   !POP S,R1; %page handle%
   !pmap();
   IF removf AND NOT tops20flag THEN !pmap(-1);
   !EXCH R1,R2; %R1 has file page handle%
   IF NOT removf AND tops20flag THEN !pmap();
      %gets page back into map%
   R2 _ R3; %access%
   RETURN;
   END.
(mapin)PROC(startpage, endpage);                                    8F
   LOCAL count, fbase, hansav;
   %get n's pages back%
      FOR count _ (fbase _ startpage) UP UNTIL >= endpage DO
         BEGIN
         IF ( hansav _ freemap[count-fbase]) # -1 THEN
            BEGIN
            R1 _ hansav.fhand;
            R3 _ 0;
            R3.hndacc _ hansav.hndacc;
            END
         ELSE
            BEGIN
            R1 _ -1;
            R3 _ 120400B6;
            END;
         R2.LH _ 4B5;
         R2.RH _ count;
         !JSYS pmap;
         END;
   %clear first cell of freemap as flag that high seg not mapped
   out%
      freemap _ 0;
   RETURN;
   END.
(hmapin)PROC;                                                       8G
   mapin( $bfree/1000B, $efree/1000B );
   RETURN;
   END.
(prgetst) PROCEDURE;                                                8H
   % returns byte pointer + count for next statement %
   LOCAL stid, sdb;
   REF sdb;
```

```
    IF (stid _ seqgen(&prseqwk)) = endfil THEN RETURN(endfil);
    IF stid.stastr THEN RETURN(stid.stpsid+1+4407B8, [stid.stpsid].L)
    ELSE
       BEGIN
       IF NOT lodprop( stid, txttyp : &sdb) THEN
          err($"No text block associated with node");
       RETURN(&sdb + sdbhdl + 4407B8, sdb.schars);
       END;
    END.

(oprchr) PROCEDURE;  % get a character for OP %                          8I
    IF opccpos = opcmax THEN RETURN(ENDCHR);
    BUMP opccpos;
    RETURN(^opcbp);
    END.

(oprnst) PROCEDURE;  % get next statement for OP %                       8J
    LOCAL stid, sdb;
    REF sdb;
    IF (stid _ seqgen(&prseqwk)) # endfil THEN
       BEGIN
       IF NOT stid.stastr THEN
          BEGIN
          IF NOT lodprop( stid, txttyp : &sdb) THEN
             err($"No text block associated with node");
          opbp _ opcbp _ &sdb + sdbhdl + 4407B8;
          opcmax _ sdb.schars + 1;
          END
       ELSE
          BEGIN
          opbp _ opcbp _ chbmty + stid.stpsid;;
          opcmax _ [stid.stpsid].L + 1;
          END;
       opccpos _ 1;
       opnewst _ TRUE;
       END;
    RETURN (stid, opbp, opcmax);
    END.

(oprtxt) %***% PROCEDURE;    % get char pos of 1st text after "name"
%                                                                        8K
    LOCAL TEXT POINTER tp;
    IF prseqwk.swcstid.stastr THEN RETURN(FALSE);
    tp _ prseqwk.swcstid;
    tp[1] _ fchtxt(tp);
    FOR tp[1] DOWN UNTIL <=1 DO
       BEGIN
       IIBP opcbp;
       BUMP opccpos;
       END;
    RETURN(opccpos);
    END.

(oprgps) PROCEDURE;                                                      8L
    RETURN (opccpos);
    END.
```

```
(oprrst) PROCEDURE   % set to passed char position %                    8N
   (pos);                                                               8M1
   IF pos = -1 THEN opccpos _ opcmax
   ELSE opccpos _ pos;
   opcbp _ chbptr(pos - 1) + opbp.RH -1;
   RETURN;
   END.

(oprlev) PROCEDURE;                                                     8N
   oplev _ prseqwk.swclvl;
   RETURN ( oplev);
   END.

(oprsvc) PROCEDURE;                                                     8O
   stvect (prseqwk.swcstid, prseqwk.swsvw);
   RETURN (prseqwk.swsvw);
   END.

(oprsig) PROCEDURE;                                                     8P
   *lit* _ NULL;
   fechsig (prseqwk.swcstid, $lit);
   RETURN ($lit);
   END.

(oprhdf) PROCEDURE;                                                     8Q
   RETURN (getfhd (prseqwk.swcstid));
   END.

(oprsid) PROCEDURE;                                                     8R
   RETURN (getsid (prseqwk.swcstid));
   END.

(oprdtc) PROCEDURE; % put date and time of creation into string %       8S
   LOCAL sdbadr;
   REF sdbadr;
   lit.L _ 0;
   IF NOT prseqwk.swcstid.stastr THEN
      BEGIN
      IF NOT lodprop( prseqwk.swcstid, txttyp :&sdbadr) THEN
         err($" No text block with this node");
      dtfrmt( sdbadr.stime, $lit );
      END;
   RETURN ($lit);
   END.

(oprdc) PROCEDURE; % put date of creation into string %                 8T
   LOCAL sdbadr;
   REF sdbadr;
   lit.L _ 0;
   IF NOT prseqwk.swcstid.stastr THEN
      BEGIN
      IF NOT lodprop( prseqwk.swcstid, txttyp :&sdbadr) THEN
         err($" No text block with this node");
      daofrmt( sdbadr.stime, $lit );
      END;
```

```
    RETURN ($lit);
    END.

(oprcid) PROCEDURE; % put ident of last editor into string %        8U
    lit.L _ 0;
    IF NOT prseqwk.swcstid.stastr THEN
       BEGIN
       idfrmt ( prseqwk.swcstid, $lit );
       END;
    RETURN ($lit);
    END.

(oprint) PROCEDURE; % return internal id format for statement %     8V
    LOCAL sdbadr, stdb;
    REF sdbadr;
    IF NOT lodprop( prseqwk.swcstid, txttyp :&sdbadr, stdb) THEN
       err($" No text block with this node");
    RETURN (getint (stdb));
    END.

(oprdt) PROCEDURE; % return internal date and time format for
statement %                                                         8W
    LOCAL sdbadr;
    REF sdbadr;
    lodprop( prseqwk.swcstid, txttyp :&sdbadr);
    RETURN (sdbadr.stime);
    END.

(optypefe) PROCEDURE % type from OP buffer to terminal via FE %     8X
    (disptyp, % window id: 1002 => print in TTY window; 1003 => CFB
    window; could also be aanother window id %
    waitflag, % 0: don't wait; 1: wait for ca; type "type OK:"; 2:
    wait for CA; no appended message. %
    clearflag, % 1 if clear before message; 0 if no clear %
    bellflag, % 0: no bell; 1: bell before; 2: bell after; 3: bell
    before and after. %
    astrng); % string/buffer to be displayed %
    typseq( disptyp, waitflag, clearflag, crno, bellflag, astrng);
    RETURN;
    END.

(opinit) PROCEDURE (da, jfn, devtyp, opflags, adgcout, comdev);     8Y
    oprwrk _ jfn;
    oprwrk[1] _ devtyp;
    oprwrk[2] _ da;
    oprwrk[3] _ $oprchr;
    oprwrk[4] _ $oprtxt;
    oprwrk[5] _ $oprrst;
    oprwrk[6] _ $oprgps;
    oprwrk[7] _ $oprlev;
    oprwrk[8] _ $oprsvc;
    oprwrk[9] _ $oprsig;
    oprwrk[10] _ $oprhdf;
    oprwrk[11] _ $oprnst;
    oprwrk[12] _ opflags;
    oprwrk[13] _ adgcout;
```

```
    oprwrk[14] _ $oprsid;
    oprwrk[15] _ comdev; % 50 = comp80, 51 = singer, 52 = videocomp %
    oprwrk[16] _ $oprdtc;
    oprwrk[17] _ $oprdc;
    oprwrk[18] _ $oprcid;
    oprwrk[19] _ $oprint;
    oprwrk[20] _ $oprdt;
    oprwrk[21] _ $optypefe;
    RETURN;
    END.

(getsbn) PROCEDURE (astrng);                                        87
    %convert string passed to sixbit name%
    LOCAL sixbit, charct, bytptr;
    REF astrng;
    bytptr _ 6B8 + $sixbit - 1;
    sixbit _ 0;
    charct _ empty;
    WHILE ((charct _ charct + 1) <= 6) AND (charct <= astrng.L) DO
      ^bytptr _ (*astrng*[charct] + 40B) .A 77B;
    RETURN(sixbit);
    END.

%.....misc. sequential file support.....%

(appseq)   % CL: ;  append string to sequential file %
PROCEDURE (jfn, astrng REF);                                        9A
    % Procedure description
      FUNCTION
        Append the string pointed to by &astrng to a 7-bit
        sequential file specified by jfn
      ARGUMENTS
        none
      RESULTS
        proc-value
      NON-STANDARD CONTROL
        none
      GLOBALS
        none
      %
    % Declarations %
    % sout it %
      !sout(jfn, &astrng + chbmty, -astrng.L);
    % Return %
      RETURN;
    END.

FINISH of SEQFIL
```