Outline for Detailed System Documentation of NLS

1 System Maintence Guide for NLS (maintence,)

- 1a Files and their contents
- 1b Steps in assembly
- 1c Steps in assembly of other features

2 Description of Special problems with NLS (special,)

- 2a Overlay structure and relabeling
  - 2a1 Overlay structure
  - 2a2 Conventions
  - 2a3 Implementation
- 2b Output overlay (MOL, META, etc.)
- 2c Formated output (pass4)
- 2d Keeping the binary code for NLS sub-systems: the C-NLS file
  - 2d1 The code problem
  - 2d2 Storage on the file and conventions
  - 2d3 The NLSDMP program
- 2e Input from keyboard
- 2f Input from QED files (pass1)
- 2g Display creation
- 2h SDB grabage collection
- 2i Errors and Aborts
- 2j Sequence generation
- 2k File link jumps

3 Other features (features,)

- 3a File clean up
- 3b Content analyzer compiler
- 3c Vectors

6h PRMSPC overlay (PRMSPC, SPCLIB)

6i STRMNP overlay (STRMNP, MNPLIB)

6j TXTEDT overlay (TXTEDT, TXTLIB)

6k MNCTRL overlay

```
:SYSGD, 10/21/69 2159:34 WHP ; (paxasis,1:xh)  (index:gebt)

1 program structure  (drawing deleted for printout)

2 for documentation on file structure see (filestructure,1)

3 for documentation on special problems see (nlsdocumentation,1)

4 for documentation on data forms see (dataforms,1)

5 (file structure) file structure

   5a for NLS file structure, see (andrews, nlsfiles, 1 :xnb)

   5b definition of terms, see (andrews, dataforms,1 :xny)

6 (OvlInd)  overlay index & debug information

   6a (auxcod)   forks and fork starting

      6a1 link to file (nls,AUXCOD, :gwJ) (kdf ERICKSON)

      6a2 starting location: orgaux=24000 page 5

      6a3 cells used:  26234

      6a4 location for temporaries: axt

      6a5 prefix for generated labels: axl

      6a6 procedures in the AUXCOD overlay

         6a6a  (qkf) freeze statement
               (AUXCOD, qkf :gwJ)

         6a6b  (qka) release frozen statement
               (AUXCOD, QKA :gwJ)

         6a6c  (qplsl) pointer show
               (AUXCOD, qplsl :gwJ)

         6a6d  (qpml) pointer delete
               (AUXCOD, qpml :gwJ)

         6a6e  (delptn) delete pointers
               (AUXCOD, delptn :gwJ)

         6a6f  (seeptr) show pointers
               (AUXCOD, seeptr :gwJ)

         6a6g  (frzst) freeze statement
               (AUXCOD, frzst :gwJ)
```

6a6h (relall) release frozen statements
(AUXCOD, relall :gwJ)

6a6i (sttus) status
(AUXCOD, sttus :gwJ)

6a6j (used) number blocks used message for status
(AUXCOD, used :gwJ)

6a6k (b1) byte 1 of a-reg
(AUXCOD, b1 :gwJ)

6a6l (b2) byte 2 of a-reg
(AUXCOD, b2 :gwJ)

6a6m (b3) byte 3 of a-reg
(AUXCOD, b3 :gwJ)

6a6n (typdat) type data
(AUXCOD, typdat :gwJ)

6a6o (inptfs) input fork - insert character intoinput buffer
(AUXCOD, inptfs :gwJ)

6a6p (inptl) input a character
(AUXCOD, inptl :gwJ)

6a6q (inptf) input fork
(AUXCOD, inptf :gwJ)

6a6r (opnffk) open file fork
(AUXCOD, opnffk :gwJ)

6a6s (tadfk) time and data fork
(AUXCOD, tadfk :gwJ)

6a6t (wtfork) alarm wait fork
(AUXCOD, wtfork :gwJ)

6a6u (chkrub :gwJ) check for rubout during insert QED
(AUXCOD, chkrub :gwJ)

6a6v (xerror) errors and aborts
(AUXCOD, xerror :gwJ)

6a6w (resetl) reset
(AUXCOD, resetl :gwJ)

6a6x (resnes) reset
(AUXCOD, resnes :gwJ)

6b (recint) recovery and initializing

6b1 link to file (nls,recint, :gwJ) (kdf MOL)

6b2 starting location: orgrin=10000 page 2

6b3 cells used: 13573

6b4 location for temporaries: rit

6b5 prefix for generated labels: ril

6b6 procedures in the RECINT overlay

6b6a (getpag) get a new page from system
(RECINT, getpag :gwJ)

6b6b (setint) initialize NLS
(RECINT, setint :gwJ)

6b6c (makro) make page read only
(RECINT, makro :gwJ)

6b6d (setnls) more initialization for NLS
(RECINT, setnls :gwJ)

6b6e (settod) todas initialization
(RECINT, settod :gwJ)

6b6f (getans) reads yes or no from keyboard
(RECINT, getans :gwJ)

6b6g (gtembk) restore pages that were dropped from PMT
(RECINT, gfembk :gwJ)

6b6h (rlssom) release some pages from PMT
(RECINT, rlssom :gwJ)

6b6i (rstart) restart for continue NLS
(RECINT, rstart :gwJ)

6b6j (setfil) set up random file
(RECINT, setfil :gwJ)

6b6k (setws) set up work station
(RECINT, setws :gwJ)

6b6l (setdis) set up display
(RECINT, setdis :gwJ)

6b6m (settos) set todas view specs
(RECINT, settos :gwJ)

6b6n (setdb) initialize dispaly buffer
(RECINT, setdb :gwJ)

```
6b6o  (setdpg) initialize data page
      (RECINT, setdpg :gwJ)

6b6p  (savpop) save pop transfer vector
      (RECINT, savpop :gwJ)

6b6q  (setrsv) zero RSVST
      (RECINT, setrsv :gwJ)

6b6r  (setsdb) zero SDBST
      (RECINT, setsdb :gwJ)

6b6s  (junkf) after reading bad file
      (RECINT, junkf :gwJ)

6b6t  (ldsbsy) load subsystem with BRS 100
      (RECINT, ldsbsy :gwJ)

6b6u  (lodrl) load relabelling for output
      (RECINT, lodrl :gwJ)

6b6v  (todpro) todas processors
      (RECINT, todpro :gwJ)

6b6w  (setps4) set up for pass 4
      (RECINT, setps4 :gwJ)

6b6x  (setprc) set up for processors
      (RECINT, setpro :gwJ)

6b6y  (runprc) run processor
      (RECINT, runprc :gwJ)

6b6z  (setmol) set up for MOL
      (RECINT, setmol :gwJ)

6b6a@  (setout) transfer to compiler
      (RECINT, setout :gwJ)

6b6aa  (bsrtn) return from process
      (RECINT, bsrtn :gwJ)

6b6ab  (waitbs) wait for left arrow
      (RECINT, waitbs :gwJ)

6b6ac  (setalm) set up alarm closk
      (RECINT, setalm :gwJ)

6b6ad  (timms) times for alarm clock
      (RECINT, timms :gwJ)

6b6ae  (setpsl) set up pass 1
      (RECINT, setpsl :gwJ)
```

6b6af (ttyck) input from TTY buffer
(RECINT, ttyck :gwJ)

6b6ag (OERGPS) return from process
(RECINT, OERGPS :gwJ)

6b6ah (stotfk) statr output fork
(RECINT, stotfk :gwJ)

6b6ai (rlsall) release pages for process
(RECINT, rlsall :gwJ)

6b6aj (rlblrf) relabel in random file blocks
(RECINT, rlblrf :gwJ)

6b6ak (actpro) activate process
(RECINT, actpro :gwJ)

6b6al (stfprv) start process fork
(RECINT, stfprv :gwJ)

6b6am (dmlink) link to (dismes :gwJ) for portal
(RECINT, dmlink :gwJ)

6c (data) data page

6c1 link to file (nls,data,:xbhjnz) (kdf ERICKSON)

6c2 starting location: orguty=14000 page 3

6c3 cells used: 17715

6d (utilty) utility package

6d1 link to file (NLS,utilty,1:gebjJ) (kdf PARSLEY)

6d2 starting location: orguty=14000 page 3

6d3 cells used: 17715

6d4 location for temporaries: utt

6d5 prefix for generated labels: utl

6d6 procedures in the UTILTY overlay

6d6a (utgd) UTILTY general declarations
(UTILTY, utgd :gwJ)

6d6b (push) push b-reg onto general stack
(UTILTY,push :gwJ)

6d6c (pop) popgeneral stack onto b-reg

(UTILTY, pop :gwJ)

6d6d (rellit) release literal register
(UTILTY, rellit :gwJ)

6d6e (getlit) get literal register
(UTILTY, getlit :gwJ)

6d6f (regadr) get register address
(UTILTY, regardr :gwJ)

6d6g (apchr) append character to a-string
(UTILTY, apchr :gwJ)

6d6h (apsr) append a-string to another
(UTILTY, apsr :gwJ)

6d6i (ldchr) load character from a-string
(UTILTY, ldchr :gwJ)

6d6j (cpysr) copy one a-string into another
(UTILTY, cpysr :gwJ)

6d6k (asrbuf) move a-string to buffer
(UTILTY, asrbuf :gwJ)

6d6l (mvbfbf) move buffer (up in core)
(UTILTY, mvbfbf :gwJ)

6d6m (mvdown) move buffer (down in core)
(UTILTY, mvdown :gwJ)

6d6n (dismes) display message
(UTILTY, dismes :gwJ)

6d6o (outmes) TODAS - output message to TTY
(UTILTY, outmes :gwJ)

6d6p (crlf) TODAS - output message to CRLF
(UTILTY, crlf :gwJ)

6d6q (err) error recovery - pop
(UTILTY, err :gwJ)

6d6r (rerror) serious error
(UTILTY, rerror :gwJ)

6d6s (abort) user error - pop
(UTILTY, abort :gwJ)

6d6t (fechcl) read characters from statement
(UTILTY, fechcl :gwJ)

```
6d6u  (getsuc) get successor of PSID
      (UTILTY, getsuc :gwJ)

6d6v  (getsub) get sub of PSID
      (UTILTY, getsub :gwJ)

6d6w  (getftl) get tail flag of PSID
      (UTILTY, getftl :gwJ)

6d6x  (getfhd) get head flag of PSID
      (UTILTY, getfhd :gwJ)

6d6y  (getsdb) get PSDB
      (UTILTY, getsdb :gwJ)

6d6z  (getssf) get word #1 of ring element
      (UTILTY, getssf :gwJ)

6d6a@  (fchsdb) fetch sdb given PSID
      (UTILTY, fchsdb :gwJ)

6d6aa  (lodrsv) load ring element given PSID
      (UTILTY, lodrsv :gwJ)

6d6ab  (storsv)  load ring  element,  set  check  sum  to  zero
      (UTILTY, storsv :gwJ)

6d6ac  (lodsdb) load sdb given PSDB
      (UTILTY, lodsdb :gwJ)

6d6ad  (getvdb) get PVDB given PSID
      (UTILTY, getvdb :gwJ)

6d6ae  (stovdb) store PUDB given PSID
      (UTILTY, stovdb :gwJ)

6d6af  (lodvdb) load VDB given PVDB
      (UTILTY, lodvdb :gwJ)

6d6ag  (lodqb) load keyword block
      (UTILTY, lodqb :gwJ)

6d6ah  (frzrfb) freeze random file block
      (UTILTY, frzrfb :gwJ)

6d6ai  (getwka) get working random file
      (UTILTY, getwka :gwJ)

6d6aj  (brs66x) delete contents of file
      (UTILTY, brs66x :gwJ)

6d6ak  (opnrff :gwJ)   open working random file
      (UTILTY, opnrff :gwJ)
```

6d6al (newrfb) get new random file block
(UTILTY, newrfb :gwJ)

6d6am (lodrfb) load random file block
(UTILTY, lodrfb :gwJ)

6d6an (rfberr) random file block error
(UTILTY, rfberr :gwJ)

6d6ao (cksum) compute check sum
(UTILTY, cksum :gwJ)

6d6ap (ovl) overlay pop
(UTILTY, ove :gwJ)

6d6aq (las) load arg on stack pop
(UTILTY, las :gwJ)

6d6ar (sap) store arg pointer pop
(UTILTY, sap :gwJ)

6d6as (laa) load arg, a only pop
(UTILTY, laa :gwJ)

6d6at (saa) store arg, a only pop
(UTILTY, saa :gwJ)

6d6au (lap) load arg pointer pop
(UTILTY, lap :gwJ)

6d6av (lsa) load subroutine arg pop
(UTILTY, lsa :gwJ)

6d6aw (sov) select overlay pop
(UTILTY, sov :gwJ)

6d6ax (sbc) subroutine call pop
(UTILTY, sbc :gwJ)

6d6ay (ovc) overlay call - no return pop
(UTILTY, ovc :gwJ)

6d6az (brv) branch to overlaypop
(UTILTY, brv :gwJ)

6d6b@ (lass) load arg string on stack pop
(UTILTY, lass :gwJ)

6d6ba (ovld) overlay
(UTILTY, ovld :gwJ)

6d6bb (ovldpa) overlay - change relabelling
(UTILTY, ovldpa :gwJ)

6d6bc (ovldn) overlay
(UTILTY, ovldn :gwJ)

6d6bd (ovlgo) to return to an overlay
(UTILTY, ovlgo :gwJ)

6d6be (ovladr) generate overlay address
(UTILTY, ovladr :gwJ)

6d6bf (ursnls) reset NLS
(UTILTY, ursnls :gwJ)

6d6bg (rubout) todas - handles rubout
(UTILTY, rubout :gwJ)

6d6bh (intr4) interrupt 4
(UTILTY, intr4 :gwJ)

6d6bi (intr5) interrupt 5 - terminate NLS
(UTILTY, intr5 :gwJ)

6d6bj (intr7 :gwJ) interrupt 7 - input fork-character
(UTILTY, intr7 :gwJ)

6d6bk (intr8) interrupt 8 - alarm clock fork
(UTILTY, intr8 :gwJ)

6d6bl intr9) interrupt 9 - pass 1 completion
(UTILTY, intr9 :gwJ)

6d6bm (intr10) interrupt 10 - input fork - rubout
(UTILTY, intr10 :gwJ)

6d6bn (hash) return hash for a-string
(UTILTY, hash :gwJ)

6d6bo (getnam) return name hash given PSID
(UTILTY, getnam :gwJ)

6d6bp (lknamh) look for name hash given in a
(UTILTY, lknamh :gwJ)

6d6bq (setpb :gwJ) reset pointer table
(UTILTY, setpb :gwJ)

6d6br (bt) branch true pop
(UTILTY, bt :gwJ)

6d6bs (bf) branch false pop
(UTILTY, bf :gwJ)

6d6bt (lai) load a immediate pop
(UTILTY, lai :gwJ)

6d6bu lbi) load b immediate pop
(UTILTY, lbi :gwJ)

6d6bv (clr) clear a state machine register pop
(UTILTY, clr :gwJ)

6d6bw (sbr) subrouting return pop
(UTILTY, sbr :gwJ)

6d6bx (atg) set abort target pop
(UTILTY, atg :gwJ)

6d6by (txt) create a new display pop
(UTILTY, txt :gwJ)

6d6bz (cec) copy entity character pop
(UTILTY, cec :gwJ)

6d6c@ (cic) copy input character pop
(UTILTY, cic :gwJ)

6d6ca (gc) get character pointed to by ab pop
(UTILTY, gc :gwJ)

6d6cb (gps) go to previous state pop
(UTILTY, gps :gwJ)

6d6cc (kset) set general flag to true pop
(UTILTY, kset :gwJ)

6d6cd (mks) mark stack for subroutine call
(UTILTY, mks :gwJ)

6d6ce (clrdpy) clear display
(UTILTY, clrdpy :gwJ)

6d6cf (clrlin) clear line
(UTILTY, clrlin :gwJ)

6d6cg (resdpy) reset display
(UTILTY, resdpy :gwJ)

6d6ch (clrall) clear entire display
(UTILTY, clrall :gwJ)

6d6ci (newabs) fid new absolute position for vectore
(UTILTY, newabs :gwJ)

6d6cj (cdsep) find lower left position of statement for Upack
(UTILTY, cdsep :gwJ)

6d6ck (compr) logical comparisons
(UTILTY, compr :gwJ)

6e (inpfbk) input feedback

6e1 link to file (nls, inpfbk, :xbjhnz) (kdf ERICKSON)

6e2 starting location: orguty=20000 page 4

6e3 cells used: 23403

6e4 location for temporaries: ift

6e5 prefix for generated labels: ifl

6e6 procedures in the INPFBK overlay

6e6a (ifgd) general declarations for input feedback (INPFBK, ifgd :gwJ)

6e6b (readbm) read bug mark (INPFBK, readbm :gwJ)

6e6c (inptc) single character input (INPFBK, inptc :gwJ)

6e6d (inptm) main programs input routine (INPFBK, inptm :gwJ)

6e6e (lkptr) look up pointer (INPFBK, lkptr :gwJ)

6e6f (gettab) todas - number spaces to next tab (INPFBK, gettab :gwJ)

6e6g (inpcuc) todas - input character, upper case (INPFBK, inpcuc :gwJ)

6e6h (inputc) todas - read character (INPFBK, inputc :gwJ)

6e6i (lookc) todas - look at next character (INPFBK, llokc :gwJ)

6e6j (outptc) todas - output character to TTY (INPFBK, outptc :gwJ)

6e6k (putchr) todas - put a character onto TTY or a-string (INPFBK, putchr :gwJ)

6e6l (rdlit) todas - read a literal from keyboard (INPFBK, rdlit :gwJ)

6e6m (resbu) todas - restore character to input buffer (INPFBK, resbu :gwJ)

6e6n (tinptc) todas - main character input routine (INPFBK, tinptc :gwJ)

6e6o (txtlit) todas - read literal into literal buffer (INPFBK, txtlit :gwJ)

6e6p (gt2lit) todas - get second literal buffer (INPFBK, gt2lit :gwJ)

6e6q (echo) todas - echo string back to TTY (INPFBK, echo :gwJ)

6e6r (echo0) todas - echo character (INPFBK, echo0 :gwJ)

6e6s (echo3) todas - echo routine (INPFBK, echo3 :gwJ)

6e6t (pushp) push ab on spec stack (INPFBK, pushp :gwJ)

6e6u (gmon) turn question mark on (INPFBK, gmon :gwJ)

6e6v (gmoff) turn question mark off (INPFBK, gmoff :gwJ)

6e6w (cflarw) position command feedback line arrow (INPFBK, cflarw :gwJ)

6e6x (san) set arrow on (INPFBK, san :gwJ)

6e6y (saf) set arrow off (INPFBK, saf :gwJ)

6e6z (ulopp) bug marks off (INPFBK, ulopp :gwJ)

6e6a@ (sbmfbc) set bug mark feedback character (INPFBK, sbmfbc :gwJ)

6e6aa (cflatc) put a-string in command feedback line (INPFBK, cflatc :gwJ)

6e6ab (ltlg) viewspecs large (INPFBK, ltlg :gwJ)

6e6ac (ltsm) viewspecs small (INPFBK, stsm :gwJ)

6e6ad (cosno) convert a-string to integer (INPFBK, cosno :gwJ)

6e6ae (asrnam) a-string to name area
(INPFBK, asrnam :gwJ)

6e6af (bkachr) back up a character in an a-string
(INPFBK, bkachr :gwJ)

6e6ag (numdpy) number display
(INPFBK, numdpy :gwJ)

6e6ah (xintrg) interrupt 9 - insert qed termination
(INPFBK, xintrq :gwJ)

6e6ai (cptstr) copy t-string to a-string
(INPFBK, cptstr :gwJ)

6e6aj (chkfrz) check frozen pages
(INPFBK, chkfrz)

6e6ak (lec) load entity character pop
(INPFBK, lec :gwJ)

6e6al (les) load entity string pop
(INPFBK, les :gwJ)

6e6am (gset) set group and entity pop
(INPFBK, gset :gwJ)

6e6an (mlfl) move literal to feedback first position - pop
(INPFBK, mlfl :gwJ)

6e6ao (mlf) move literal to feedback, next position
(INPFBK, mlf :gwJ)

6e6ap (mlfr) move literal to feedback, replace last
(INPFBK, mlfr :gwJ)

6e6aq (rep) repeat in case statement pop
(INPFBK, rep :gwJ)

6e6ar (dfs) define state pop
(INPFBK, dfs :gwJ)

6e6as (cta) copy t-string to a-string pop
(INPFBK, cta :gwJ)

6e6at (creg) append char in c to reg no. in a - anypop
(INPFBK, creg :gwJ)

6e6au (breg) append character in b to reg no. in a - anypop
(INPFBK, breg :gwJ)

6e6av (sreg) append sar in b to reg no. in a - anypop
(INPFBK, sreg :gwJ)

6e6aw (sba) set bug to arrow - anypop
(INPFBK, sba :gwJ)

6e6ax (stp) set but to plus - anypop
(INPFBK, sbp :gwJ)

6e6ay (ta) display in text area - anypop
(INPFBK, ta :gwJ)

6e6az (mrf) move register to feedback line, next
(INPFBK, mrf :gwJ)

6e6b@ (mrfr) move register to feedback line, replace
(INPFBK, mrfr :gwJ)

6e6ba (mrkl) mark feedback line, first position
(INPFBK, mrkl :gwJ)

6e6bb (mrk) mark feedback line, current position
(INPFBK, mrk :gwJ)

6e6bc (kin) input a character - anypop
(INPFBK, kin :gwJ)

6e6bd (kct) character test anypop
(INPFBK, kct :gwJ)

6e6be (pbm) process bug mark anypop
(INPFBK, pbm :gwJ)

6e6bf (cse) begin case statement anypop
(INPFBK, cse :gwJ)

6e6bg (spcb) spec bug mark anypop
(INPFBK, spcb :gwJ)

6e6bh (spcr) spec a register anypop
(INPFBK, spcr :gwJ)

6e6bi (mrs) move register to string anypop
(INPFBK, mrs :gwJ)

6e6bj (bkc) backspace character inan a-string
(INPFBK, bkc :gwJ)

6e6bk (dpn) display in name area
(INPFBK, dpn :gwJ)

6e6bl (spcn) spec hask of a-string
(INPFBK, spcn :gwJ)

6e6bm (cct) char. class test
(INPFBK, cct :gwJ)

6e6bn (zero) zero the spec stack pointer
(INPFBK, zero :gwJ)

6e6bo (zap) general reset
(INPFBK, zap :gwJ)

6e6bp (sapsp) general reset
(INPFBK, sapsp :gwJ)

6e6bq (feedlt) feed view specs to (setlt :gwJ)
(INPFBK, feedlt :gwJ)

6e6br (tch) test character pop
(INPFBK, tch :gwJ)

6f (mnctrl) main control overlay

6f1 link to file (nls, mnctrl, :xbbjhnz) (kdf ERICKSON)

6f2 starting location: orguty=24000 page 5

6f3 cells used: 27450

6f4 procedures in the MNCTRL overlay

6f4a (wait) wait for MNCTRL
(MNCTRL, wait :gwJ)

6f4b (caqm) command accept, question mark
(MNCTRL, caqm :gwJ)

6f4c (dmani) display, then go to main
(MNCTRL, dmain :gwJ)

6f4d setdum) set up dummy statement
(MNCTRL, setdum :gwJ)

6f4e (cmdrst) command reset
(MNCTRL, cmdrst :gwJ)

6f4f (mdispec) kludge for "set"
(MNCTRL, mdispec :gwJ)

6f4g (main) read a character
(MNCTRL, main :gwJ)

6f4h (wc) what character
(MNCTRL, wc :gwJ)

6g (prmspc) parameter specification overlay

6g1 link to file (nls, prmspc, :xbjhnz) (kdf MOL)

6g2 starting location: orguty=30000 page 6

6g3 cells used: 33767

6g4 procedures in the PRMSPC overlay

6g4a (recre) recreate display
(PRMSPC, recre :gwJ)

6g4b (nuview) recreate display starting at new position
(PRMSPC, nuview :gwJ)

6g4c (setvsp) set view specs
(PRMSPC, setvsp :gwJ)

6g4d (specbug) spec a bug mark
(PRMSPC, specbug :gwJ)

6g4e (waitp) wait in prwspc
(PRMSPC, waitp :gwJ)

6g4f (molrt) output mo2
(PRMSPC, molrt :gwJ)

6g4g (buglspec) spec a bug mark
(PRMSPC, buglspec :gwJ)

6g4h (bug2spec) spec two bug marks
(PRMSPC, buglspec :gwJ)

6g4i (bug3spec) spec three bug marks
(PRMSPC, bug3spec :gwJ)

6g4j (bugltspec) spec a bug and read literal
(PRMSPC, butltspec :gwJ)

6g4k (bug2tspec) spec two bugs and read literal
(PRMSPC, bug2tspec :gwJ)

6g4l (atext) read literal
(PRMSPC, atext :gwJ)

6g4m (integer) read an integer from keyboard
(PRMSPC, atext :gwJ)

6g4n (ltspec) read view specs from keyboard
(PRMSPC, ltspec :gwJ)

6g4o (statno) read statement number from keyboard
(PRMSPC, statno :gwJ)

6g4p (statnam) read statement name from keyboard
(PRMSPC, statnam :gwJ)

```
6g4q  (specn) spec name
      (PRMSPC, specn :gwJ)

6g4r  (jpspcn :gwJ)   jump spec name
      (PRMSPC, jpspcn :gwJ)

6g4s  (jpsplt) jump lt
      (PRMSPC, jpsplt :gwJ)

6g4t  (kgtwd) get word for keyword
      (PRMSPC, kgtwd :gwJ)

6g4u  (specit) spec a bug or string
      (PRMSPC, specit :gwJ)

6g4v  (qfs) jump to item
      (PRMSPC, qfs :gwJ)

6g4w  (qjo) jump to origin
      (PRMSPC, qjo :gwJ)

6g4x  (qhs) jump to name
      (PRMSPC, qhs :gwJ)

6g4y  (qjb) jump back
      (PRMSPC, qjb :gwJ)

6g4z  (qjv) jump vector label
      (PRMSPC, qjv :gwJ)

6g4a@  (qjp) jump predecessor
      (PRMSPC, qjp :gwJ)

6g4aa  (qjs) jump successor
      (PRMSPC, qjs :gwJ)

6g4ab  (qju) jump up
      (PRMSPC, qju :gwJ)

6g4ac  (qjd) jump down
      (PRMSPC, qjd :gwJ)

6g4ad  (qjh) jump to head
      (PRMSPC, qjh :gwJ)

6g4ae  (qjt) jumpt o tail
      (PRMSPC, qjt :gwJ)

6g4af  (qib) insert branch
      (PRMSPC, qib :gwJ)

6g4ag  (qis) insert statement
      (PRMSPC, qis :gwJ)
```

6g4ah (qrb) replace branch
(PRMSPC, qrb :gwJ)

6g4ai (qrg) replace group
(PRMSPC, qrg :gwJ)

6g4aj (qrp) replace plex
(PRMSPC, qrp :gwJ)

6g4ak (atexti) literal for insert
(PRMSPC, atexti :gwJ)

6g4al (atextr) literal for replace
(PRMSPC, atextr :gwJ)

6g4am (adj) adjust level
(PRMSPC, adj :gwJ)

6g4an (setlt) set viewspecs
(PRMSPC, setlt :gwJ)

6g4ao (enwdpy) number display for view specs
(PRMSPC, enwdpy :gwJ)

6g4ap (aplit) append literal
(PRMSPC, aplit :gwJ)

6g4aq (aplita) literal input routine
(PRMSPC, aplita :gwJ)

6g4ar (terlin) terminate line
(PRMSPC, terlin :gwJ)

6g4as (nterch) enter character inliteral
(PRMSPC, nterch :gwJ)

6g4at (frzsav) save view specs for frozen
(PRMSPC, frzsav :gwJ)

6g4au (dpych) display character subroutine file
(PRMSPC, dpych :gwJ)

6g4av (jspush) push on jump stack
(PRMSPC, jspush :gwJ)

6g4aw (jpsh) push word on jump stack
(PRMSPC, jpsh :gwJ)

6g4ax (savlt) create word to save view specs
(PRMSPC, savlt :gwJ)

6g4ay (fechux)
(PRMSPC, fechux :gwJ)

6g4az (gtlit2) get second literal register
(PRMSPC, gtlit2 :gwJ)

6g4b@ (softld) soft command delete
(PRMSPC, softld :gwJ)

6g4ba (pbug) process bug mark
(PRMSPC, pbug :gwJ)

6g4bb (fndchr) find character given line, column numbers
(PRMSPC, fndchr :gwJ)

6h (vctedt) special characters and vector libe

6h1 link to file (nls, vcted, :xbjhnz) (kdf MOL)

6h2 starting location: orguty=24000 page 5

6h3 cells used: 26107

6h4 procedures in the VCTEDT overlay

6h4a (jpspco) jump vector label
(VCTEDT, jpspco :gwJ)

6h4b (qcd) copy drawing
(VCTEDT, qcd :gwJ)

6h4c (qdd) delete drawing
(VCTEDT, qdd :gwJ)

6h4d (qmd) move drawing
(VCTEDT, qmd :gwJ)

6h4e (vwait) wait for VCTEDT
(VCTEDT, vwait :gwJ)

6h4f (new vdb) get new vector data block
(VCTEDT, newvdb :gwJ)

6h4g (nwvdb :gwJ) get new vector data block
(VCTEDT, nwvdb :gwJ)

6h4h (fndlin) find line given bug mark
(VCTEDT, fndlin :gwJ)

6h4i (sqrt) square root routine
(VCTEDT, sqrt :gwJ)

6h4j (dsq) distance squared between two points
(VCTEDT, dsq :gwJ)

6h4k (namend) mark end points of line

(VCTEDT, namend :gwJ)

6h4l (endset) set up names for endpoints of line
(VCTEDT, endset :gwJ)

6h4m (endput) record which end of line selected
(VCTEDT, endput :gwJ)

6h4n (switch) change selection to other end of line
(VCTEDT, switch :gwJ)

6h4o (labasr) move label to a-string
(VCTEDT, labasr :gwJ)

6h4p (butchr) read character from buffer
(VCTEDT,butchr :gwJ)

6h4q (fnd lab) find label given bug mark
(VCTEDT, fndlab :gwJ)

6h4r (distsq) distance squared to label
(VCTEDT, distsq :gwJ)

6h4s (buglab) attach old label to bug
(VCTEDT, buglab :gwJ)

6h4t (spclab) spec label number
(VCTEDT, spclab :gwJ)

6h4u (to bug) set up bug for label
(VCTEDT, tobug :gwJ)

6h4v (labres) restore moved label
(VCTEDT, labres :gwJ)

6h4w (movlab) move label
(VCTEDT, movlab :gwJ)

6h4x (dellab) deete label
(VCTEDT, dellab :gwJ)

6h4y (newlab) new label to bug
(VCTEDT, newlab :gwJ)

6h4z (putlab) fix new label
(VCTEDT, putlab :gwJ)

6h4a@ (mcheck) used by copy and move drawing
(VCTEDT, mcheck :gwJ)

6h4aa (mcd) copy drawing
(VCTEDT, mcd :gwJ)

6h4ab (mmd) move drawing
(VCTEDT, mmd :gwJ)

6h4ac (mdd) delete drawing
(VCTEDT, mdd :gwJ)

6h4ad (mrkpos :gwJ) mark position of selection
(VCTEDT, mrkpos :gwJ)

6h4ae (mrkoff) turn mark off
(VCTEDT, mrkoff :gwJ)

6h4af (flwdx) convert X-position to full word
(VCTEDT, flwdx :gwJ)

6h4ag (flwdy) convert Y-position to full word
(VCTEDT, flwdy :gwJ)

6h4ah (vreset) reset for vector package
(VCTEDT, vreset :gwJ)

6h4ai (grdfix) force bug mark onto grid
(VCTEDT, grdfix :gwJ)

6i (vctrl) vector package

6i1 link to file (nls, vctrl, :xbjhnz) (kdf MOL)

6i2 starting location: orguty=30000 page 6

6i3 cells used: 33167

6i4 procedures in the VCTRL overlay

6i4a (vmain) main control for vectors
(VCTRL, vmain :gwJ)

6i4b (bdot) turn bug into a dot
(VCTRL, bdot :gwJ)

6i4c (offon) switch ofr spacing
(VCTRL, offon :gwJ)

6i4d (onoff) switch for spacing
(VCTRL, onoff :gwJ)

6i4e (lab) read a label
(VCTRL, lab :gwJ)

6i4f (endput) select an endpoint
(VCTRL, endput :gwJ)

6i4g (vbugl) get a bug select for vector

(VCTRL, vbug1 :gwJ)

6i4h (vbug2) get 2 bugs select for vector
(VCTRL, vbug2 :gwJ)

6i4i (vbug1) get a bug select for label
(VCTRL, vbug1 :gwJ)

6i4j (vcwait) wait for vpact
(VCTRL, vcwait :gwJ)

6i4k (labget) select a label
(VCTRL, labget :gwJ)

6i4l (vcted) initialization for vpack
(VCTRL, vcted :gwJ)

6i4m (vctfin) termination for vpack
(VCTRL, vctfin :gwJ)

6i4n (chkold) check old VDB
(VCTRL, chkold :gwJ)

6i4o (vabort) abort from vpack
(VCTRL, vabort :gwJ)

6i4p (delet) delete a vector
(VCTRL, delet :gwJ)

6i4q (trans) translate a vector
(VCTRL, trans :gwJ)

6i4r (vertln) make vector vertical
(VCTRL, vertln :gwJ)

6i4s (horzln) make vector horizontal
(VCTRL, horzln :gwJ)

6i4t (newlin) create new line from endpoints
(VCTRL, newlin :gwJ)

6i4u (putlin) put new line in drawing
(VCTRL, putlin :gwJ)

6i4v (put1ln) put first new line of sequence
(VCTRL, putlin :gwJ)

6i4w (put12n) put later lines in sequence
(VCTRL, put12n :gwJ)

6i4x (movend) move endpoint of line
(VCTRL, movend :gwJ)

6i4y :gwJ) movdrw) move drawing
(VCTRL, movdrw :gwJ)

6i4z (delpic) delete picture
(VCTRL, delpic :gwJ)

6i4a@ (grdset) turn grid on?off
(VCTRL, grdset :gwJ)

6i4aa (grdsiz) change size of grid
(VCTRL, grdsiz)

6i4ab (grdmak) make a grid
(VCTRL, grdmak :gwJ)

6i4ac (spcnum) specify a number
(VCTRL, spcnum :gwJ)

6j (diddl) "diddle" (viewcontrol)

6j1 link to file (NLS, diddl, :xbjhnz) (kdf ERICKSON)

6j2 starting location: orguty=24000 page 5

6j3 cells used: 27045

6j4 procedures in the DIDDL overlay

6j4a (diiddx) main control for view change
(DIDDL, diiddx :gwJ)

6j4b (pwait) wait in DIDDL
(DIDDL, pwait :gwJ)

6j4c (on-off) switch in DIDDL
(DIDDL, on-off :gwJ)

6j4d (intg) read integer in DIDDL
(DIDDL, intg :gwJ)

6j4e (prmset) parameter set in DIDDL
(DIDDL, prmset :gwJ)

6j4f (char) read a character in DIDDL
(DIDDL, char :gwJ)

6j4g (numasr) convert number to a-string
(DIDDL, numasr :gwJ)

6j4h (tabset) initialize to set tabs
(DIDDL, tabset :gwJ)

6j4i (tabput) put waqy tab value

(DIDDL, tabput :gwJ)

6j4j (acget) get current parameter word for armed cursor
(DIDDL, acget :gwJ)

6j4k (acput) store parameter word for armed cursor
(DIDDL, acput :gwJ)

6j4l (csave) save character
(DIDDL, csave :gwJ)

6j4m (nmdlt) set first name delimiter
(DIDDL, nmdlt :gwJ)

6j4n (nmdl2) set second name delimiter
(DIDDL, nmdl2 :gwJ)

6j4o (mrkget) get bug mark parameters
(DIDDL, mrkget :gwJ)

6j4p (mrkput) put bug mark parameters
(DIDDL, mrkput :gwJ)

6j4q (mrcput) set bug mark character
(DIDDL, mrcput :gwJ)

6j4r (cflget) get cfl parameters
(DIDDL, cflget :gwJ)

6j4s (cflput) put cfl parameters
(DIDDL, cflput :gwJ)

6j4t (cflapt) put cfl arrow parameters
(DIDDL, cflapt :gwJ)

6j4u (dtput) put date and time parameters
(DIDDL, dtput :gwJ)

6j4v (ecoput) put echo reg parameters
(DIDDL, ecoput :gwJ)

6j4w (namput) put name reg parameters
(DIDDL, namput :gwJ)

6j4x (tstput) put test area parameters
(DIDDL, tstput :gwJ)

6j4y (vsaput) put view spec area parameters
(DIDDL, vsaput :gwJ)

6j4z (colput) put number of columns
(DIDDL, colput :gwJ)

6j4a@ (shocol) show number of columns
(DIDDL, shocol :gwJ)

6j4aa (indput) put indenting
(DIDDL, indput :gwJ)

6j4ab (shoind) show indenting
(DIDDL, shoind :gwJ)

6j4ac (vrtput) put vertical increment
(DIDDL, vrtput :gwJ)

6j4ad (vrtptl) update vertical increment in display
(DIDDL, vrtptl :gwJ)

6j4ae (shovrt) show vertical increment
(DIDDL, shovrt :gwJ)

6j4af (linput) put number of linew
(DIDDL, linput :gwJ)

6j4ag (sholin) show number of lines
(DIDDL, sholin :gwJ)

6j4ah (mrkpos :gwJ) mark position of bug selection
(DIDDL, mrkpos :gwJ)

6j4ai (mhps) mark typed position
(DIDDL, mhps :gwJ)

6j4aj (isave) save parameters
(DIDDL, isave :gwJ)

6j4ak (ivestr) restore parameters
(DIDDL, ivestr :gwJ)

6j4al (etrstr) entry for setlt
(DIDDL, etrstr :gwJ)

6j4am (mobck) move buffer
(DIDDL, mobck :gwJ)

6j4an (setsize) set size of characters
(DIDDL, setsiz)

6j4ao (sethic) set horizontal increment
(DIDDL, sethic :gwJ)

6j4ap (setbld) set bold face
(DIDDL, setbld :gwJ)

6j4aq (setflc) set flicker
(DIDDL, setflc :gwJ)

6j4ar (setitl) set italics
(DIDDL, setitl :gwJ)

6j4as (setdsp) set display on?off
(DIDDL, setdsp :gwJ)

6k (keywd) keyword overlay

6k1 link to file (NLS, keywd, :xbjhnz) (kdf ??)

6k2 starting location: orguty=24000 page 5

6k3 cells used: 25335

6k4 procedures in the KEYWD overlay

6k4a (kmain) main control for keyword
(KEYWD, kmain :gwJ)

6k4b (kwait) wait - in KEYWD
(KEYWD, kwait :gwJ)

6k4c (kyex) keyword execute
(KEYWD, kyex :gwJ)

6k4d (kyfndh) find place in table for keyword select
(KEYWD, kyfndh :gwJ)

6k4e (kyputw) put weight for keyword
(KEYWD, kyputw :gwJ)

6k4f (kyfgt) forget a keyword
(KEYWD, kyfgt :gwJ)

6k4g (kyfgth) forget a keyword
(KEYWD, kyfgth :gwJ)

6k4h (kyfgta) forget all keywords
(KEYWD, kyfgta :gwJ)

6k4i (kylist) keyword list
(KEYWD, kylist :gwJ)

6k4j (kychr) buffer control for KEYWD
(KEYWD, kychr :gwJ)

6k4k (kyxtrn) XTRNAM for keyword
(KEYWD, kyxtrn :gwJ)

6k4l (numasr) number to a-string
(KEYWD, numasr :gwJ)

6l (lnksub) links and substitute

6l1 link to file (NLS, lnksub, :xbjhnz) (kdf ERICKSON)

6l2 starting location: orguty=20000 page 4

6l3 cells used: 21125

6l4 procedures in the LNKSUB overlay

6l4a (hlnkdm) link delimiter
(LNKSUB, hlnkdm :gwJ)

6l4b (jlpon) jump link possible statement name
(LNKSUB, jlpon :gwJ)

6l4c jlpfn) jump link possible file name
(LNKSUB, jlpfn :gwJ)

6l4d (jlpun) jump link possible user name
(LNKSUB, jlpun :gwJ)

6l4e (subst :gwJ) substitute
(LNKSUB, subst :gwJ)

6l4f (subfch) get character for substitute
(LNKSUB, subfch :gwJ)

6l4g (getch) get character for substitute
(LNKSUB, getch :gwJ)

6l4h (subsx) initialize substitution
(LNKSUB, subsx :gwJ)

6l4i (subint) initialize substitution
(LNKSUB, subint :gwJ)

6m (strmnp) structure manipulation overlay

6m1 link to file (nls, strmnp, :xbjhnz) (kdf MOL)

6m2 starting location: orguty=20000 page 4

6m3 cells used: 23210

6m4 procedures in the STRMNP overlay

6m4a (qds) delete statement
(STRMNP, qds :gwJ)

6m4b (qdb) delete branch
(STRMNP, qdb :gwJ)

6m4c (qdg) delete group
(STRMNP, qdg :gwJ)

6m4d (qdp) delete plex
(STRMNP, qdp :gwJ)

6m4e (qcs) copy statement
(STRMNP, qcs :gwJ)

6m4f (qcb) copy branch
(STRMNP, qcb :gwJ)

6m4g (qcg) copy group
(STRMNP, qcq :gwJ)

6m4h (qcp) copy plex
(STRMNP, qcp :gwJ)

6m4i (qms) move statement
(STRMNP, qms :gwJ)

6m4j (qmb) move branch
(STRMNP, qmb :gwJ)

6m4k (qmp) move plex
(STRMNP, qmp :gwJ)

6m4l (qmg) move group
(STRMNP, qmg :gwJ)

6m4m (rpllit) replace with literal
(STRMNP, rpllit :gwJ)

6m4n (qrs) replace statement
(STRMNP, qrs :gwJ)

6m4o (inslit) insert literal
(STRMNP, inslit :gwJ)

6m4p (qbs) break statement
(STRMNP, qbs :gwJ)

6m4q (abj) join statement
(STRMNP, qbj :gwJ)

6m4r (qkr) release frozen
(STRMNP, qkr :gwJ)

6m4s (qxb) substitute branch
(STRMNP, qxb :gwJ)

6m4t (qxs) substitute statement
(STRMNP, qxs :gwJ)

6m4u (qxp) substitute plex
(STRMNP, qxp :gwJ)

6m4v (qxg) substitute group
(STRMNP, qxg :gwJ)

6m4w (grptst) group test
(STRMNP, grptst :gwJ)

6m4x (remgrp) remove group
(STRMNP, remgrp :gwJ)

6m4y (setgrp) setgroup
(STRMNP, setgrp :gwJ)

6m4z (plxset) set plex
(STRMNP, plxset :gwJ)

6m4a@ (insgrp) insert group
(STRMNP, insgrp :gwJ)

6m4aa (inss) insert statement as successor
(STRMNP, inss :gwJ)

6m4ab (insd) insert statement down
(STRMNP, insd :gwJ)

6m4ac (delgrp) delete group
(STRMNP, delgrp :gwJ)

6m4ad (movtst) structure move test
(STRMNP, movtst :gwJ)

6m4ae (subgrp) substitute in group
(STRMNP, subgrp :gwJ)

6m4af (copgrp) copy group
(STRMNP, copgrp :gwJ)

6m4ag (tailx) find tail of plex
(STRMNP, tailx :gwJ)

6m4ah (sourcx) find source of plex
(STRMNP, sourcx :gwJ)

6m4ai (headx) find head of plex
(STRMNP, headx :gwJ)

6m4aj (pred) find predecessor of ring element
(STRMNP, pred :gwJ)

6m4ak (subtst) test for substructure
(STRMNP, subtst :gwJ)

6m4al (sstftl) set tail flag
(STRMNP, sstftl :gwJ)

6m4am (sstfhd) set head flag
(STRMNP, sstfhd :gwJ)

6m4an (resftl) reset fail flag
(STRMNP, resftl :gwJ)

6m4ao (resfhd) reset tail flag
(STRMNP, resfhd :gwJ)

6m4ap (setsuc) set successor
(STRMNP, setsuc :gwJ)

6m4aq (set sub) set sub
(STRMNP, setsub :gwJ)

6m4ar (pusha) push a-reg on general stack
(STRMNP, pusha :gwJ)

6m4as (atc) compare pop
(STRMNP, atc :gwJ)

6m4at (stosuc) store successor
(STRMNP, stosuc :gwJ)

6m4au (stosub) store sub
(STRMNP, stosub :gwJ)

6m4av (stofhd) store head flag
(STRMNP, stofhd :gwJ)

6m4aw (stoftl) store tail flag
(STRMNP, stoftl :gwJ)

6m4ax (stossf) store word 1 of ring element
(STRMNP, stossf :gwJ)

6m4ay (intrsr) create initial ring element
(STRMNP, intrsr :gwJ)

6m4az (newrsr) new ring element
(STRMNP, newrsr :gwJ)

6m4b@ (nwrsub) new ring element
(STRMNP, nwrsub :gwJ)

6m4ba (frersr) put ring element on free list
(STRMNP, frersr :gwJ)

6m4bb (frevdb) free vector data block
(STRMNP, frevdp :gwJ)

6m4bc (relst) release statement form frozen list
(STRMNP, relsf :gwJ)

6n (clnup) file clean up

6n1 link to file (nls, clnup, :xbjhnz) (kdf ERICKSON)

6n2 starting location: orguty=24000 page 5

6n3 cells used: 27725

6n4 procedures in the CLNUP overlay

6n4a (clmain) entry for file cleanup
(CLNUP, clmain :gwJ)

6n4b (fckfin) success exit from file check
(CLNUP, fckfin :gwJ)

6n4c (fckbad) failure exit file check
(CLNUP, fckbad :gwJ)

6n4d (crsu) check ring blocks
(CLNUP, crsu :gwJ)

6n4e (csdb) check SDB'S
(CLNUP, csdb :gwJ)

6n4f (cvdb) check VDB'S
(CLNUP, cvdb :gwJ)

6n4g (cqb) check keyword block
(CLNUP, cqb :gwJ)

6n4h (ckstrc) check structure of file
(CLNUP, ckstrc :gwJ)

6n4i (patch) patch break in ring
(CLNUP, patch :gwJ)

6n4j (fr) free a lost element
(CLNUP, fr :gwJ)

6n4k (fstftl) store tail flag
(CLNUP, fstftl :gwJ)

6n4l (fstfhd) store head flag
(CLNUP, fstfhd :gwJ)

6n4m (fstnam) store name
(CLNUP, fstnam :gwJ)

6n4n (legali) check for legal PSID
(CLNUP, legali :gwJ)

6n4o (legald) check for legal PSDB

```
      (CLNUP, legald :gwJ)

6n4p  (legalv) check for legal PVDB
      (CLNUP, legalv :gwJ)

6n4q  (blksav)  try  to  save  a  bad  random  file  block
      (CLNUP, blksav :gwJ)

6n4r  (gfrfb) get a rfb for CLNUP
      (CLNUP, gfrfb :gwJ)

6n4s  (fldstb :gwJ)  lod sdb
      (CLNUP, fldstb :gwJ)

6n4t  (fldvdb) load VDB
      (CLNUP, fldvdb :gwJ)

6n4u  (gdsdb) check SDB
      (CLNUP, gdsdb :gwJ)

6n4v  (chkdb) check data blocks for statement
      (CLNUP, chkdb :gwJ)

6n4w  (getchr) read character from ADB
      (CLNUP, getchr :gwJ)

6n4x  (badchr) replace bad character in SDB
      (CLNUP, badchr :gwJ)

6n4y  (upit) update pointer for CLNUP
      (CLNUP, upit :gwJ)

6n4z  (updasp) update pointer for CLNUP
      (CLNUP, updasp :gwJ)

6n4a@  (fchlst) fetch lost elements
      (CLNUP, fchlst :gwJ)

6n4aa  (dump) free lost elements
      (CLNUP, dump :gwJ)

6n4ab  (chkpnt) check pointers
      (CLNUP, chkpnt :gwJ)

6n4ac  (crnch) crunch pointer table
      (CLNUP, crnch :gwJ)

6n4ad  (fxnam) copy of extra name for CLNUP
      (CLNUP, fxnam :gwJ)

6n4ae  (nuhash)  generate  new  hash's  for  all  statements
      (CLNUP, nuhash :gwJ)
```

6n4af (outres) output results of CLNUP
(CLNUP, outres :gwJ)

6o (pass1) qed input

6o1 link to file (nls, pass1, :xbjhnz) (kdf MOL)

6o2 starting location: orguty= page

6o3 cells used:

6p (ioctl) i/o control

6p1 link to file (nls, ioctl, :xbjhnz) (kdf ERICKSON)

6p2 starting location: orguty=24000 page 5

6p3 cells used: 27762

6p4 procedures in the IOCTL overlay

6p5 % free core and save blocks %

6p5a (fixun) fix user name at start of file name
(IOCTL, fixun :gwJ)

6p5b (frecor) write core blocks on random file
(IOCTL, frecor :gwJ)

6p5c (frcrff) close current, open working file
(IOCTL, frcrff :gwJ)

6p5d (copfil) copy file
(IOCTL, copfil :gwJ)

6p5e (wrtblk) write file block
(IOCTL, wrtblk :gwJ)

6p5f (copblk) copy file block from input to output
(IOCTL, copblk :gwJ)

6p6 % open files %

6p6a (rdhdr) read header block of file
(IOCTL, rdhdr :gwJ)

6p6b (opnckp) open checkpoint file
(IOCTL, opnckp :gwJ)

6p6c (opnfil) open file
(IOCTL, opnfil :gwJ)

6p7 %jump file stack %

```
    6p7a  (pushno) push current view on file stack
          (IOCTL, pushno :gwJ)

    6p7b  (pophs) pop an element from file stack
          (IOCTL, pophs :gwJ)

    6p7c  (bumpho) read nest file stack element
          (IOCTL, bumpho :gwJ)

    6p7d  (rtnhs) read file stack entry
          (IOCTL, rtnhs :gwJ)

    6p7e  (stkchk) remove word from file stack
          (IOCTL, stkchk :gwJ)

    6p7f  (pshhs) push word on file stack
          (IOCTL, pshhs :gwJ)

    6p7g  (pphs) pop word from file stack
          (IOCTL, pphs :gwJ)

6p8  % save and restore viewspecs %

    6p8a  (isavlt) create word to save viewspecs
          (IOCTL, isavlt :gwJ)

    6p8b  (ireslt) restore view specs
          (IOCTL, ireslt :gwJ)

6p9  % jump stuff %

    6p9a  (clrjs) clear jump stack
          (IOCTL, clrjs :gwJ)

    6p9b  (fusern) set file user name
          (IOCTL, fusern :gwJ)

    6p9c  (savrff) save working copy data
          (IOCTL, savrff :gwJ)

    6p9d  (ofdate) open file date
          (IOCTL, ofdate :gwJ)

    6p9e  (fnroot) get file name from root
          (IOCTL, fnroot :gwJ)

    6p9f  (lnktyp) determine link type
          (IOCTL, lnktyp :gwJ)

    6p9g  (lnknul) determine if null link
          (IOCTL, lnknul :gwJ)

    6p9h  (pushjs) push on intrafile jump stack
```

(IOCTL, pushjs :gwJ)

6p9i (popjs) pop intrafile jump stack
(IOCTL, popjs :gwJ)

6p9j (bumpjs) read next jump stack element
(IOCTL, bumpjs :gwJ)

6p9k (pshjs) push word on intrafile jump stack
(IOCTL, pshjs :gwJ)

6p9l (ppjs) pop word from intrafile jump stack
(IOCTL, ppjs :gwJ)

6p10 % spl for IOCTL %

6p10a (fwait) wait for IOCTL
(IOCTL, fwait :gwJ)

6p10b (fjwait) jump wait
(IOCTL, fjwait :gwJ)

6p10c (qlq) load file or checkpoint
(IOCTL, qlq :gwJ)

6p10d (qlf) load file
(IOCTL, qlf :gwJ)

6p10e (qlc) load checkpoint
(IOCTL, qlc :gwJ)

6p10f (opns3i) open type 3 symbolic file or input
(IOCTL, opns3i :gwJ)

6p10g (rstfil) open current file and recreate display
(IOCTL, rstfil :gwJ)

6p10h (ofstn2) open file, name given in STN 2*
(IOCTL, ofstn2 :gwJ)

6p10i (qoq) main output control
(IOCTL, qoq :gwJ)

6p10j (opns3) open symbolic type 3 file for output
(IOCTL, opns3 :gwJ)

6p10k (opnfo) open random file for output
(IOCTL, opnfo :gwJ)

6p10l (qof) output file
(IOCTL, qof :gwJ)

6p10m (qoc) output checkpoint

(IOCTL, qoc :gwJ)

6p10n (spcdpy) spec display for link routines
(IOCTL, spcdpy :gwJ)

6p10o (sethfn) set jump file name
(IOCTL, sethfn :gwJ)

6p10p (qbl) jump link
(IOCTL, qbl :gwJ)

6p10q (qjj) jump return
(IOCTL, qjj :gwJ)

6p10r (qja) jump ahead
(IOCTL, qja :gwJ)

6p10s (qbq) jump file
(IOCTL, qbq :gwJ)

6p10t (qaq) alarm clock
(IOCTL, qaq :gwJ)

6p11 % insert QED and pass 4 %

6p11a %insert QED and pass 4 %

6p11a1 (stpas4) start pass 4
(IOCTL, stpas4 :gwJ)

6p11a2 (pslcse) convert case for pass 1
(IOCTL, pslcse :gwJ)

6p11a3 (qiq) insert QED branch
(IOCTL, qiq :gwJ)

6p11b % portal %

6p11b1 (cap) activate portal processor
(IOCTL, cap :gwJ)

6q (seqgen) seqgence generator overlay

6q1 link to file (NLS, seqgen, :xbjhnz) (kdf MOL)

6q2 starting location: orguty=30000 page 6

6q3 cells used: 32263

6q4 procedures in the SEQGEN overlay

6q4a (fcp) fix character position - CA pop
(SEQGEN fcp :gwJ)

6q4b (lc) logical complement - CA pop
(SEQGEN, lc :gwJ)

6q4c (pcp) push character position - CA pop
(SEQGEN, pcp :gwJ)

6q4d (ieq) initials equal - CA pop
(SEQGEN, leq :gwJ)

6q4e (ine) initials not equal - CA pop
(SEQGEN,lne :gwJ)

6q4f (snc) date since - CA pop
(SEQGEN, snc :gwJ)

6q4g (bfr) date before - CA pop
(SEQGEN, bfr :gwJ)

6q4h (rst :gwJ) reset flag - CA pop
(SEQGEN, rst :gwJ)

6q4i (scp) set character position - CA pop
(SEQGEN, scp :gwJ)

6q4j (atst) string test, addressin A - CA pop
(SEQGEN, stst :gwJ)

6q4k (atstf) string test, address in A, floating CA pop
(SEQGEN, atstf :gwJ)

6q4l (tst) string test, string in code, - CA pop
(SEQGEN, tst :gwJ)

6q4m (tstf) string test, string in code, floating - CA pop
(SEQGEN, tstf :gwJ)

6q4n (tstr) test string - CA
(SEQGEN, tstr :gwJ)

6q4o (tstfr) test string, floating - CA
(SEQGEN, tstfr :gwJ)

6q4p (gcft) get character for test - CA
(SEQGEN, gcft :gwJ)

6q4q (arb) arbitrary number - CA pop
(SEQGEN, arb :gwJ)

6q4r (bfs) branch false and scan - CA pop
(SEQGEN, bfs :gwJ)

6q4s (brc) branch on repeat count - CA pop
(SEQGEN, brc :gwJ)

6q4t (ccp) compare character positions - CA pop
(SEQGEN, ccp :gwJ)

6q4u (ccv) character class value - CA pop
(SEQGEN, ccv :gwJ)

6q4v (pps) pop push down stack - CA pop
(SEQGEN, pps :gwJ)

6q4w (sd) set direction - CA pop
(SEQGEN, sd :gwJ)

6q4x (cpf) character position function - CA pop
(SEQGEN, cpf :gwJ)

6q4y (pdc) pointer decrement - CA pop
(SEQGEN, pdc :gwJ)

6q4z (fechsl) initialization for sequence generator
(SEQGEN, fechsl :gwJ)

6q4a@ (fechsn) fetch next statement - sequence generator
(SEQGEN, fechsn :gwJ)

6q4aa (seqset) set up for sequence generator
(SEQGEN, seqset :gwJ)

6q4ab (seqnxt) get next PSID in sequence
(SEQGEN, seqnxt :gwJ)

6q4ac (seqpat) check statement for pattern match
(SEQGEN, seqpat :gwJ)

6q4ad (getint) get initials from SDB
(SEQGEN, getint :gwJ)

6q4ae (gettim) get time from SDB
(SEQGEN, gettim :gwJ)

6q4af (getdat) get date from SDB
(SEQGEN, getdat :gwJ)

6q4ag (stoflg) store flag for pattern match
(SEQGEN, stoflg :gwJ)

6q4ah (schtxt) fetch 6th word of SDB leader
(SEQGEN, schtxt :gwJ)

6q4ai (kyfch) keyword sequence generator
(SEQGEN, kyfch :gwJ)

6q4aj (stvect) generate statement number vector
(SEQGEN, stvect :gwJ)

6q4ak (stvctp) statement vector for successor
(SEQGEN, stvctp :gwJ)

6q4al (stpos) position of ring element in its plex
(SEQGEN, stpos :gwJ)

6q4am (fechno) fetch statement number of given PSID
(SEQGEN, fechno :gwJ)

6q4an (fechnp) fetch statement number of successor
(SEQGEN, fechnp :gwJ)

6q4ao (fechnm) statement number given statement vector
(SEQGEN, fechnm :gwJ)

6q4ap (setseq) initialize sequence generator - area
(SEQGEN, setseq :gwJ)

6r (sdbmnp) sdb manipulator

6r1 link to file (nls, sdbmnp, :xbjhnz) (kdf MOL)

6r2 starting location: orguty=24000 page 5

6r3 cells used: 26024

6r4 procedures in the SDBMNP overlay

6r4a (newsdb) make a new SDB
(SDBMNP, newsdb :gwJ)

6r4b (endsdb) termination of creating SDB
(SDBMNP, endsdb :gwJ)

6r4c (xtrnam) extract name from SDB
(SDBMNP, xtrnam :gwJ)

6r4d (comsr) compare strings
(SDBMNP, comsr :gwJ)

6r4e (ncis) number characters in string
(SDBMNP, ncis :gwJ)

6r4f (stonam) store name hash in ring element
(SDBMNP, stonam :gwJ)

6r4g (stosdp) store PSDB in ring element
(SDBMNP, stosdp :gwJ)

6r4h (delst) delete statement
(SDBMNP, delst :gwJ)

6r4i (fresdb) put SDB on free list

(SDBMNP, fresdb :gwJ)

6r4j (fndrom) find room for new SDB
(SDBMNP, fndrom :gwJ)

6r4k (isroom) looks for adequate room for SDB
(SDBMNP, isroom :gwJ)

6r4l (gcolx) garbage collection of SDB'S
(SDBMNP, gcolx :gwJ)

6r4m (gcol) garbage collection of SDB'S
(SDBMNP, gcol :gwJ)

6r4n (cmpfr) file compactor
(SDBMNP, cmpfr :gwJ)

6s (txtedt) text edit

6s1 link to file (nls, txtedt, :xbjhnz) (kdf MOL)

6s2 starting location: orguty=34000 page 7

6s3 cells used: 37464

6s4 procedures in the TXTEDT overlay

6s4a (recred) recreate display
(TXTEDT, recred :gwJ)

6s4b (subs) substitute
(TXTEDT, subs :gwJ)

6s4c (subi) initialization for substitute
(TXTEDT, subi :gwJ)

6s4d (gdmys) set dummy statement
(TXTEDT, gdmys :gwJ)

6s4e (setrot) set root statement
(TXTEDT, setrot :gwJ)

6s4f (sttxt) set text
(TXTEDT, sttxt :gwJ)

6s4g (brkst) break statement
(TXTEDT, brkst :gwJ)

6s4h (staptx) join statement
(TXTEDT, staptx :gwJ)

6s4i (stlit) create statement from literal
(TXTEDT, stlit :gwJ)

```
6s4j  (cdlim) character delimiter
      (TXTEDT, cdlim :gwJ)

6s4k  (idr) invisibledelimiter routine
      (TXTEDT, idr :gwJ)

6s4l  (tdr) text delimiter routine
      (TXTEDT, tdr :gwJ)

6s4m  (ndr) number delimiter routine
      (TXTEDT, ndr :gwJ)

6s4n  (wdr) word delimiter routine
      (TXTEDT, wdr :gwJ)

6s4o  (wdr2) another word delimiter routine
      (TXTEDT, wdr2 :gwJ)

6s4p  (vdr) visible delimiter routine
      (TXTEDT, vdr :gwJ)

6s4q  (deltx) delete pointers in statement
      (TXTEDT, deltx :gwJ)

6s4r  (cpchtx) copy character
      (TXTEDT, cpchtx :gwJ)

6s4s  (cshft) case shift
      (TXTEDT, cshft :gwJ)

6s4t  (del) delete text
      (TXTEDT, del :gwJ)

6s4u  (mrchtx) move character
      (TXTEDT, mrchtx :gwJ)

6s4v  (mvwdvs) move word, visible
      (TXTEDT, mvwdvs :gwJ)

6s4w  (rpl) replace text
      (TXTEDT, rpl :gwJ)

6s4x  (cpwdvs) copy word, visible
      (TXTEDT, wpwdvs :gwJ)

6s4y  (qcc) copy character
      (TXTEDT, qcc :gwJ)

6s4z  (qcw) copy word
      (TXTEDT, qcw :gwJ)

6s4a@  (qcn) copy number
      (TXTEDT, qcn :gwJ)
```

```
6s4aa  (qci) copy invisible
      (TXTEDT, qci :gwJ)

6s4ab  (qcv) copy visible
      (TXTEDT, qcv :gwJ)

6s4ac  (qct) copy text
      (TXTEDT, qct :gwJ)

6s4ad  (qdc) delete character
      (TXTEDT, qdc :gwJ)

6s4ae  (qdw) delete word
      (TXTEDT, qdw :gwJ)

6s4af  (qdn) delete number
      (TXTEDT, qdn :gwJ)

6s4ag  (qdi) delete invisible
      (TXTEDT, qdi :gwJ)

6s4ah  (qdv) delete visible
      (TXTEDT, qdv :gwJ)

6s4ai  (qdt) delete text
      (TXTEDT, qdt :gwJ)

6s4aj  (qic) insert character
      (TXTEDT, qic :gwJ)

6s4ak  (qiw) insert word
      (TXTEDT, qiw :gwJ)

6s4al  (qin) insert number
      (TXTEDT, qin :gwJ)

6s4am  (qii) insert invisible
      (TXTEDT, qii :gwJ)

6s4an  (qiv) insert visible
      (TXTEDT, qiv :gwJ)

6s4ao  (qit) insert text
      (TXTEDT, qit :gwJ)

6s4ap  (qmc) move character
      (TXTEDT, qmc :gwJ)

6s4aq  (qmw) move word
      (TXTEDT, qmw :gwJ)

6s4ar  (qmn) move number
      (TXTEDT, qmn :gwJ)
```

6s4as (qmi) move invisible
(TXTEDT, qmi :gwJ)

6s4at (qmv) move visible
(TXTEDT, qmv :gwJ)

6s4au (qmt) move text
(TXTEDT, qmt :gwJ)

6s4av (qrc) replact text
(TXTEDT, qrc :gwJ)

6s4aw (qrw) replace word
(TXTEDT, qrw :gwJ)

6s4ax (qrn) replace number
(TXTEDT, qrn :gwJ)

6s4ay (qri) replace invisible
(TXTEDT, qri :gwJ)

6s4az (qrv) replace visible
(TXTEDT, qrv :gwJ)

6s4b@ (qrt) replace text
(TXTEDT, qrt :gwJ)

6s4ba (qsc) set character
(TXTEDT, qsc :gwJ)

6s4bb (qsw) set word
(TXTEDT, qsw :gwJ)

6s4bc (qsn) set number
(TXTEDT, qsn :gwJ)

6s4bd (qsi) set invisible
(TXTEDT, qsi :gwJ)

6s4be (qsv) set visible
(TXTEDT, qsv :gwJ)

6s4bf (qst) set text
(TXTEDT, qst :gwJ)

6s4bg (qss) set statement
(TXTEDT, qss :gwJ)

6s4bh (qpf) pointer fix
(TXTEDT, qpf :gwJ)

6s4bi (qprs) pointer delete statement
(TXTEDT, qprs :gwJ)

6s4bj (qprt) pointer delete text
(TXTEDT, qprt :gwJ)

6s4bk (qprw) pointer delete word
(TXTEDT, qprw :gwJ)

6s4bl (fixptr) pointer fixup
(TXTEDT, fixptr :gwJ)

6s4bm (delptr) delete pointer
(TXTEDT, delptr :gwJ)

6s4bn (insptr) insert pointer
(TXTEDT, insptr :gwJ)

6s4bo (apachr) append a character
(TXTEDT, apachr :gwJ)

6s4bp (apastr) append a-string
(TXTEDT, apastr :gwJ)

6s4bq (aptstr) append t-string
(TXTEDT, aptstr :gwJ)

6s4br (bsc) begin statement construction - pop
(TXTEDT, bsc :gwJ)

6s4bs (esc) end statement construction - pop
(TXTEDT, esc :gwJ)

6s4bt (cpp) fixup pointers and append t-string - pop
(TXTEDT, cpp :gwJ)

6s4bu (kps) append a-string - pop
(TXTEDT, kps :gwJ)

6s4bv (kpr) append a register - pop
(TXTEDT, kpr :gwJ)

6s4bw (dlp) delete pointers - pop
(TXTEDT, dlp :gwJ)

6s4bx (cpw) append t-string - pop
(TXTEDT, cpw :gwJ)

6s4by (pfx) pointer fix - pop
(TXTEDT, pfx :gwJ)

6t (cacmpl) content-analyzer compiler

6t1 link to file (nls, cacmpl, :xbjhnz) (kdf ERICKSON)

6t2 starting location: orguty=34000 page 7

6t3 cells used: 37311

6u (outovl) output overlay

6u1 link to file (nls, ~~procs~~ outov:jhnzw) (kdf MOL) ✓

6u2 starting location: orguty=34000 page 7

6u3 cells used:

6v (mol) mol compiler ✓

6v1 link to file (paxton, nmolr:jhnzw)

6v2 starting location: orguty=0 page 0

6v3 cells used:

6w (spl) spl compiler

6w1 link to file (andrews, ??:jhnzw) ⊗

6w2 starting location: orguty=0 page 0

6w3 cells used:

6x (meta) tree meta compiler

6x1 link to file (andrews, ~~nmeta~~ tree:jhnzw)

6x2 starting location: orguty=0 page 0

6x3 cells used:

6y (pass4) formatted hard copy

6y1 link to file (, :jhnzw) ⊗

6y2 starting location:

6y3 cells used:

6z (disbuf) display buffer

6z1 link to file (andrews, ~~??~~ disbuf:jhnzw) (kdf ER)

6z2 starting location: orguty=34000 page 7

6z3 cells used: 37715

6a@ (cdsply) create display

6a@1 link to file (nls, cdsply, :xbjhnz) (kdf ERICKSON)

6a@2 starting location: orguty=24000 page 0

6a@3 cells used: 27630

6a@4 procedures in the CDSPLY overlay

6a@4a (credis) main for create display (CDSPLY, credis :gwJ)

6a@4b (cdvect) create vector display (CDSPLY, cdvect :gwJ)

7 (Index) categories for NLS procedure

7a file handling

7a1 (fileio) RANDOM FILE I/O
* (lodrfb) (rdhdr)

7a2 (filcopy) FILE COPYING
* (copfil) (getwka)

7a3 (filref) FILE BLOCK REFERENCING
* (lodrsv) (lodrfb) (lodsdb)

7a4 (filcontrol) FILE CONTROL
* (opnffk) (lodrfb) (brs66x) (rdhdr)

7a5 (statusblocks) FILE STATUS BLOCK MAINTENCE
* (lodrsv) (lodsdb) (lodrfb) (lodvdb)

7a6 (workingcopy) WORKING COPY OF FILE
* (getwka) (lodrfb) (copfil) (opnrff)

7b display

7b1 (messages) MESSAGES TO USER
* (err) (dismes) (rerror) (abort)

7b2 (aborts) JUST THAT
* (err) (dismes) (abort) (rerror)

7b3 (text-display) ROUTINES TO PUT UP FILE TEXT
* (credis) (cdvect)

7b4 (changedis) CHANGING DISPLAY PARAMETERS
* (diddl)

7c input/output

7c1 (literalinput) reading keyboard input * (getlit)

7c2 (input) work station input * (inptc) (inptfs) (inptf)

7d initialization

7e structure manipulation

7f text manipulation

7g graphics

8 (Bugs) list of bugs

8a unanchored scans do not do an SCP at the start, and thus fail on certain multiple concats.

8b to restore a non-existent view crashes nls

8c the content analyzer does not reset the flags in a fie you are jumping to, this results in the wrong statementes beign shown.

8d the frozen machinery does not clear its list on a jump link, resulting in ugly psids beign used to create the display.

8e Jump to end does not call pushjs (save the display position for jump return, ahead)

8f fix ES so that cd doesn't blank the screen

8g rubout sometimes does not work right: only one gets you back to the exec after OC. INPTRF not always getting reset??

8h Also, a JUMP LINK bug creates an illegal entity abort occationally if one end of the link is the first or last character of the statement.

8i Doing a JUMP FILE with a name or number in the link which does not exist in the file results in an abort -- with the screen unchanged even though the jump to the file did take place. Do a JUMP TO ORIGIN if this happens.

8j There is a bug in the statement inserting level adjustment. If lots of U's are typed, the statement number offered is incorrect -- then if the statement is inserted and a center dot is typed to finish the insert, a fatal error occurs. (This is because the SUC of 0 is smashed).

8j1 This often happens automatically on INsert qed branch if the origin statement (which is without a statement number) is left on the from of the file being inserted.

8k On literal type-in echoing, there is still a wraparound bug. The last character of a line sometimes is displayed on top of the first.

8l There are some problems with the display not being cleared properly if the user aborts out of a literal type-in which is longer than the text on the screen.

8m The literal type-in is erased is an abort occurs during type-in. For example, if the alarm clock goes off.

8n The first line looses its indenting after an abort out of a literal type-in.

8o File clean-up results in a fatal error when the file has certain kinds of errors in it.

8p When typing in a file name, the user name part is not command recognized. But then everyone has grown used to this. RIGHT?

8q Character selecting in a statement with special characters (italics, underline, blinking, etc) is often wrong and can cause a fatal error.

8r Aborting out of a jump command after making a view change changes the view parameters but does not change the screen to match -- it just aborts. On the next display re-creation, the new view parameters are used.

8s Aborting out of a LOAD FILE at view specifying time leaves the user looking at his old file, but with his new file loaded. Do a JUMP TO ORIGIN.

8t The display of a literal type-in shifts to the right one character when a backspace character or word is typed, if the entity is word or visable. This is because there is a space automatically inserted on the front which is not diplayed to begin with, but is displayed after a backspace of some kind.

9 (map) of symbols and binaries

9a Symbols: all bands have utilty, data, and recint

9a1 176

9a1a inpfbk, vctedt, vcted2, disbuf

9a2 177

9a2a inpfbk, clnup, diddl, keywd, disbuf

9a3 178

9a3a auxcod, inpfbk, mnctrl, prmspc, disbuf

9a4 179

9a4a strmnp

9a5 180

9a5a seqgen, sdbmnp, txtedt, txtedt2

9a6 181

9a6a seqgen, cdsply, disbuf

9b Binaries

9b1 :nls

9b1a prmspc, sdbmnp, recint, utilty, inpfbk, mnctrl, seqgen, txtedt

9b2 :1NLS

9b2a ioctl, diddl, keywd, auxcod, strmnp, cdsply, clnup, 2txtedt

9b3 :2NLS

9b3a vctedt, 2vctedt, cacmpl

10 thoughts for improvements

10a trails

10a1 Trail returns can be done through a push down stack. Every time the seq-gen makes a trail branch it pushes the psid on a stack. The user may define a trail return syntax. If he has one, and it is found in a statement, the stack is poped and the poped psid used to start the seq-gen out again.

10b content analyzer

10b1 Coroutines also seem to be the answer to the content analyzer case problem. A set of standard coroutines could be selectively invoked to modify the text on the way to the TST and CV tests. They would convert cases, skip punction, etc. They could be invoked and dis-invoked independent of the paren structure of the pattern, a la directions.

10b2 Don't have the sequence gen. set the content flags in the ring unless there is a working copy. If the user is looking at a real file, do it the hard way.

10b2a WATCHIT:

10b2b Don't reset the bits when the pattern is compiled unless a working copy is being used

10b2c Don't even look at the bits in the seq. gen. unless a working copy is being used

10b2d when a woking copy is created, if pattern is on, reset all of the ring bits for pattern.

10c editing

10c1 Coroutines should be used for APTSTR, APCHR, and APASR. There will be a set of them, they check for directives, do case shifts, do text substitues, check for names, etc.

10d entity selecting (specing)

10d1 Make JUMP TO NAME work (FIRST & NEXT).

10d1a This involves a new way to spec names, etc.

10d2 Just put character pointer on spec stack

10d2a or name hash

10d2b or integer

10d2c or screen position

10d3 There is a problem with file jumps, when name or number does not exist - need to recreate the display so that it corresponds to the file. I think this is a specing problem.

10d4 A similar problem exists when a continue is done and the file cannot be opened - screen is not correct.

10e create display

10e1 Create display should have lots of smarts. All kinds of flags will be set for it to find out what happened. Consider the following:

10e1a When first called the whole display may have to be recreated, so check:

10e1a1 A new file loaded.

10e1a2 text not up there (e.g. the tree is being displayed)

10e1b The sequence of statements

10e1b1 needs to be recreated if

10e1b1a A new display start

10e1b1b sequencing view parameter change

10e1b1b1 these are ca and trail

10e1b2 needs to be checked if

10e1b2a A new statement has been put in (insert or copy)

10e1b2b Statement moved

10e1b2c Statement deleted

10e1b2d sequencing view parameter change such as

10e1b2d1 level and branch only

10e1c The formatting of statements

10e1c1 needs to be changed if

10e1c1a formatting view specs have changed

10e1c1a1 these are st. nos, names

10e1c2 needs to be checked if

10e1c2a These view specs have changed

10e1c2a1 blank line, pointers, indenting, clip, blank line

10e1c2b a truncation change

10e1d a statement by statement check needs to be made, for:

10e1d1 Text changed

10e1d2 Pointers changed

10e1d3 Vector part changed

10e2 implementation scheme

10e2a create display is called with parameters-- and puts up text of one file in a given screen area -- somebody above him knows what is going on over the whole screen.

10e2b character bugging may be done with this scheme:

10e2b1 the mode within a given area text is uniform

10e2b2 each given area is implicitly divided into characer position and each position numbered in a well defined manner.

10e2b3 A table gives the PSID and character count thusly:

10e2b3a A table entry contains a T-pointer, and two character counts.

10e2b3b The character counts are character positions within the display area.

10e2b3c The table is sorted on starting character counts.

10e2b3d A table element means that the text between the two character counts starts at the given T-pointer.

10e2b3e At bug select time, a pointer to this table and the character count is put on the spec stack.

10e2c for putting text up in a hurry

10e2c1 move the text from the SDB a word at a time - perhaps doing a shift. - or put in nulls so a fast loop can be used.

10e2c2 To make this work:

10e2c2a tab and cr should be special characters.

10e2c2b also, a bit should be set in the ring to indicate that special characters are in the text.

10e2c2c statement-insert in a statement has that bit set

10e2c2d Of course, we need upper and lower case display hardware.

10e2c2e If the special character bit is on, do it the hard way.

10e2c3 Better have some kind of check to know when the line is full -- then back up to a word gap.

10e2d basic routine heirachry:

10e2d1 an overloard takes care of screen orgainzation and selects area for frozen statements, perhaps.

10e2d1a cdsply fills up a text area.

10e2d1a1 the sequence gen plays its usual role

10e2d1a2 A statement routine calls the seq. gen. and puts up the blank line or initials, number, sets the indenting, etc.

10e2d1a2a A low level routine fills a buffer for one line.

10e2d1a2b Maybe a fast guy and a smart guy, for special characters.

10e2d1a3 somewhere in here a routine takes care of tabular data branches.

10e2d1a4 And graphic pictures

10e3 classification of display areas

10e3a zero dimensional areas:

10e3a1 are:

10e3a1a the armed cursor

10e3a1b the disarmed cursor

10e3a1c the bug mark

10e3a2 are subject to:

10e3a2a size

10e3a2b mode (intensity, filcker, italics)

10e3b one dimensional areas:

10e3b1 are:

10e3b1a echo reg.

10e3b1b message area

10e3b1c name area

10e3b1d date-time

10e3b1e command feedback line

10e3b1f viewspec(s) (one for each file area)

10e3b2 are subject to all of the above and:

10e3b2a position (this may be relative -??)

10e3b2b horizontal increment

10e3b2c columns

10e3c two dimensional areas

10e3c1 are:

10e3c1a the literal area

10e3c2 are subject to (in addition to the above)

10e3c2a vertical increment

10e3c2b rows

10e3c2c we may want to specify a boundary rather than giving rows, columns, vert. and horiz. increments.

10e3d file areas

10e3d1 are:

10e3d1a there are several file areas. They will make the frozen area unnecessary.

10e3d2 are subject to (in addition to above):

10e3d2a names

10e3d2b numbers (ugh)

10e3d2c indenting

10e3d2d truncation

10e3d2e blank line

10e3d2f pointers

10e3d2g tree

10e3d2h initials

10e3d2i picture clip

10e3d2j statement-insert

10e3d2k relative indenting (with plex only)

10e3d2l vowels

10e3d2m ? special characters if the vectors are ever good??

10e3d3 are modified (filtered) by:

10e3d3a content analyzer

10e3d3b branch only

10e3d3c plex only

10e3d3d sub-file

10e3d3e level

10e3d4 the content (sequence) is determined by:

10e3d4a list (normal)

10e3d4b trail

10e3d4c keyword

10e3e things we forgot:

10e3e1 pointers

10e3e2 vector spacing

10e3e3 vector scaling

10e4 other thoughts

10e4a need control over parameter for literal area, frozen area, message area.

10e4b want to have up to four views on screen --- up to two files opened for a user. Maybe two file views for each of two users collobrating, or one for four users...

10e4c want to take a way of describing view in text -- both view set and viewchange text strings.

10e4d incorperate view setting and diddle stuff in same command - allow diddle commands in view spec part of links??

10e4e Want to define a sequence of views from a text string - then jump from one to the next on a ca.

10f external core usage & display

10f1 Can all of the display buffer information fit in ADATA, with the buffers in external core?

10f2 Between now and external core time, simulate it with POPs or something. - have UTILTY code to relabel the frozen page in and move a buffer into it.

10f3 Of course, rewrite create display and figure out the network business

10f3a MNCTRL part of it is rather clear.

10f3b But how will PRMSPC part of it work?

10g file IO

10g1 Aviod creating statment numbers at all costs. Perhaps produce relative position numbers instead. Indenting by credis creates a problem (but not when branch only is on). Need to keep the level or vector for the statement at top of screen.

10g2 We can cut down on the amount of file IO by

10g2a Having copfil write out the blocks from core rather than reading all blocks and copying them.

10g2b Have frecor avoid calling getwka unless it has to write a block. We would also need a word in the file header to indicate that it was changed to make this work. ???

10g2c Have a usage table for core blocks - have LODRFB write out the block used least since the last write. - This table maintained by the LOD routines (LODSDB, LODRSV, LODVDB, etc).

10g3 The readability of opnfil can be improved if it is rewritten to get rid of all its labels and GOTOs.

10h efficient use of the data page

10h1 A longer input buffer should be used when PASS1 is running

10h2 Perhaps it will be necessary to have another RW page to contain seldom referrenced informatio. Such as file link stack.

10h3 How to make all kinds of room in ADATA:

10h3a Perhaps use cells 20b to 77b?

10h3b Rewriting the overlay stuff will make the RW table quite small (one word for each overlay).

10h3c Putting out better SPL code may cut down the string area needed.

10h3c1 The case save block may be eliminated.

10h3d All non-writable cells should be moved to UTILTY.

10h3e All lowest level subroutines can share temps since they call no one. This is true even in UTILTY.

10h3f The KHAR block for PASS1 could be eliminated by having PASS1 obtain a RF core block. Or Pass1 could fit in a RW page from C-NLS, and have it's own very large buffer.

10h3g How about having the routines that do overlay calls and need to save their return location do it on their own -- then the data page can be squeezed rather simply.

10h3h ICORTB can be in UTITLTY or better yet, in RECINT.

10h3i Can the CA code be put in a RF block?? Just how would it work if the pattern went with the file?

10h3j Some file parameters are unnecessary.

10h4 Give content analyzer a 1K core block for RW storage.

10i more effecient spl code

10i1 Overhaul the input-feedback code produced by the compiler.

10i1a Put out more dense code that simplifies the pops.

10i1b Use string addresses, not register numbers.

10i2 Make the syntax for overlay calls the same as the link syntax.

10j new features

10j1 Have OPNFIL accecpt a ? after file name, to type a different name.

10j2 A smart RDHDR procedure will be needed when the file header format changes considerably. Have it update an old kind of header.

10j3 Try to code the universal status block loading procedure;

10j4 Have code to initialize the data page and disbuf in RECINT

10j5 Write the merge file stuff. -- figure out the multiple file buisness.

10j6 Put in JUMP TO CONTEXT VISIBLE, TEXT and WORD.

10j7 Fix up some miscellaneous stuff, namely:

10j7a put view spec wait in JL, JF-, JR,JA

10j7b Fix alarm clock overflow problem

10j7c Don't create a statement number unless it is absolutely necessary

10j7d EXECUTE LINK SHOW (file links, ahead and return)

10j7e change IDENTITY to ITEM

10j8 LOAD PROGRAM and OUTPUT PROGRAM commands.

10j9 EXECUTE FINISHED and LOGOUT commands.

10j10 Put in the file branch select (The real branch only).

10j11 Have Jr and Ja return to previous jump command

10j12 Collect statistics on number of file IO's per DFS. This varies a great deal with kind of operations. Need a way to control it to get meaningful numbers.

10k Wild ideas:

10k1 have the ability to get another page for RF blocks (jimmy up rfifcbx and crpgad) for things like file cleanup, gcol and maybe create display.

10k2 have a "continue" copy of a compiler around. Would need to load it, use it, and dump it to a file for later use. The idea is to save the symbol table.

10k3 Give PASS4 a page to scratch in so that it won't use fechcl to run up and down the statement.

10k4 Could we have PASS4 go directly to the printer - using the display buffer page, or external core. This would spread out the compute load considerably.

1 NLS Random file structure and handling /NLSFILES

1a General description

1a1 The major design considerations for NLS files determined the present format and structure.

1a1a It is desirable to have virtually no limit on the size of a file. This means that it is not practical to have an entire file in core when viewing or working on it.

1a1b A goal in the design was to have the time involved to carry out operations on a file not be linear with the length of a file, but remain constant as a function of the length of a file. That is, small operations on a large file should take the same time as on a small file. In this way the user and the system do not pay a penalty for large files.

1a1c The system had to include graphic statements, and perhaps other forms of data, as well as text.

1a2 As a result of these considerations, a random file scheme was chosen. The file is logically divided into 1K blocks. There are several different types of blocks, and each type has its own structure, since it contains a different type of data.

1a3 The structure of a file is entirly separate from the data that makes up the contents of the statement. Different types of blocks (i.e. not structure blocks) contain the contents. Some blocks contain information that is part of the file, but does not belong to the structure.

1a4 The following size limitations were placed on files in general:

1a4a The maximum number of statements is 2032. (This is 2048 minus 2 RSV elements for each of eight structure blocks).

1a4b The textual part of a statement is limited to about 3000 characters. Special characters (boldface, underline, italics, etc.) count as two characters.

1a4c Any other data asociated with a statement is limited to a maximum of 1K file block (for example a picture).

1a5 Several limitations are parameters to the system at system assembly time. These are currently set as follows.

1a5a The structure of a file is limited to 13 levels in depth.

1a5b A statement name is limited to 30 characters.

1a5c The total number of text blocks allowed is 55.

1a5d The maximum number of pointers is 10.

1a5e The total number of random file blocks cannot exceed 64.

1a5f The maximum number of vector data blocks is 1.

1b File Block structure

1b1 The header block

1b1a In each file, there is a header block that contains information about the particular file.

1b1b The header block remains in memory while an NLS user is working with the file.

1b1c The contents of the header block:

1b1c1 (FCREDT) The file creation date.

1b1c2 (FUNO) The file owner's user number. This is the NLS user that created the file.

1b1c3 (NLSVWD) A secret word that indicates the header format. At present there is only one header format in use.

1b1c4 (HEDSZ) THe number of words in the file header block.

1b1c5 (FINIT) The initials of the user that last wrote the file.

1b1c6 (LWDAT) THe date at the last writing.

1b1c7 (LWTIM) The time at the last writing (in system format- see brs 39).

1b1c8 (NAMDL1) THe left name delimiter character, in the rightmost byte.

1b1c9 (NAMDL2) And the right name delimiter.

1b1c10 (RALS) THe running average length of statement in characters.

1b1c11 (TNSG) THe total number of statements generated in the life of this file.

1b1c12 (RFBS) The random file block status.

1b1c12a (rfbstl) The number of words in the RFBS table.

1b1c12b Each word of the RFBS table corresponds to a random file block, and indicates the status of that block. The file header is file block zero. The number in the RFBS entry means:

1b1c12b1 (zero) the block is not allocated, and does not exist.

1b1c12b2 (positive) The block is allocated, and is in memory rather than on the IO device, and the positive number is the actual starting memory location for the block.

1b1c12b3 (minus one) The block is allocated, but has not been initialized.

1b1c12b4 (negative) The block is not in core. The number is the negative of the used word count. In the case of text blocks, -2 indicates that the block contains no garbage SDB's, and need not be garbage collected.

1b1c13 (SDBST) The SDB status block table indicates the status of each SDB type block. The value of an SDBST entry means:

1b1c13a (zero) The block is not allocated.

1b1c13b (non-zero) The value gives the block number, that is, the entry into RFBS for that block.

1b1c14 (SDBSTL) the number of entries in SDBST.

1b1c15 (RSVST) The RSV status block table indicates the status of each RSV type block. The values have the same meaning as for SDBST.

1b1c16 (RSVSTL) The number of entries in RSVST.

1b2 File block format

1b2a Each random file block has a eight word header. This header contains:

1b2a1 (0) The checksum of the block.

1b2a2 (1) The used word count (always greater than the header size).

1b2a3 (2) The block type, to indicate

1b2a3a (0) The header block

1b2a3b (1) an SDB block

1b2a3c (2) The query block

1b2a3d (3) an RSV block

1b2a3e (4) A vector block

1b2a4 (3) In the case of

1b2a4a Rsv blocks, the free list pointer

1b2a4b SDB blocks, the grabage collection flag (=-1 if collected).

1b2a5 (4) The status table entry number (not RFBS entry, but SDBST or RSVST entry, or etc.)

1b2a6 (5-7) not used

1b3 RSV blocks

1b3a RSV blocks contain the structure of the file. Each block contains the structure of a maximum of 254 statements. There can be up to eight RSV blocks in a file, but not all need be allocated.

1b3b The block is broken into 254 four word RSV elements, each potentially describes the structural location of a statement.

1b3c Each four word RSV element in an allocated block is either in the structure of the file, or it is on the free list for the block it is in. Free lists consists of a chain of pointers, starting with the third word of the RSV block header, and ending with a zero pointer. A pointer here, is a relative address based on zero for the first word of the block. The pointers are in the first word of the four word element, and the other three words are zero. See (inpfbk, newrsv).

1b3d A RSV element is ponted to by a PSID. The 3 high order bits give the RSV block number (entry into the RSVST table), and the 8 low order bits of a PSID indicate the location within the block, approximately.

1b3d1 This is approximate because PSID's start at zero, and the first eight words of each block contain the header. Thus, before breaking the PSID, a displacement is added (RSVHDR), which is two. Then the high order 3 bits are the RSV block number, and the low order 8 bits, when multiplied by 4, give the relative address of the element within the block.

1b3e Every file has at least one RSV element in the structure. That is the origin statement and is always PSID zero. THat is, it's RSV element is the first element in RSV block zero.

1b3f The contents of a RSV element are described in (dataforms, ring element).

1b4 SDB blocks

1b4a A type one random file block is made up of SDB (statement data blocks). A SDB contains the text for a statement.

1b4b An SDB is a variable sized block of words with a six word header.

1b4b1 The header contains:

1b4b1a (0) 00600000b plus the number of words in the SDB. The sign bit is on if this SDB is garbage.

1b4b1b (1) The PSID of this statement.

1b4b1c (2) The date this copy of this statement was created.

1b4b1d (3) and the time

1b4b1e (4) and initials.

1b4b1f (5) The number of character in this statement. This includes the ENDCHR's ?

1b4b1g (6) An integer which indicates the first character that is not part of the name of this statement.

1b4c The words following the header contains the text of the statement, three characters per word. The text includes an end character (ENDCHR, code 377b) on each end of the statement. The last word is filled to a word boundry with ENDCHR's. An SDB is always an even number of words long, hence the last word may not be used.

1b4d The characters in a statement are explicitly numbered, with the first ENDCHR being number zero. A two word entity consisting of a PSID and a character count is a T-pointer, and indicates a particular character within the file. See (dataforms, t-pointer).

1b5 VBD blocks

1b6 Others

1b6a (query)

1b6b (spchr)

1c Handling

1c1 Core tables, IO

1c1a The random files are read into core by blocks. Two pages in NLS (RFBP1 and RFBP2 overlay pages) are logically divided into four 1K core blocks to contain the file blocks. Thus, up to four file blocks may be in core at a time. When a file block is requested, if all four are in use, one block will be written out. Core blocks may be "frozen" in, however, so that they will not be changed.

1c1b The file block IO is handled by (utilty, lodrfb:g).

1c1c (RFIFCB) The random file index for core blocks indicates which file block is in each core block. A zero indicates that no file block is there, a positive number gives the random file block number (index to RFBS).

1c1d (FRZCPT) The frozen core block table indicates which of the core blocks have been frozen. "frozen" indicates to the file block loading procedure (LODRFB) that that core block must not be changed.

1c1e (CRPGAD) the core block address table gives the actual core aress for each core block.

1c1f (RFICBX) The core blocks are numbered from zero to RFICBX, currently three.

1c2 File copy see (ioctl, copfil:g).

:AHIDS, 07/03/69 1105:29 MGC ;

1 AHI DATA STRUCTURE

1a A.M. 276 CLASS NOTES

1b Compliments of the HYPERTEXT EDITING SYSTEM ?

1b1 CENTER FOR COMPUTER & INFORMATION SCIENCES

1b2 BROWN UNIVERSITY

1b3 PROVIDENCE, RHODE ISLAND

1b4 30 March, 1969

$\}$?

2 1. %DATA %STRUCTURE

2a The AHI data structure has as its basic unit the "statement." The statement is the smallest textual unit defined, and is simply a textual string. The file (i.e., collection of statements) is hierarchically oriented in a tree structure, each statement being a node in the tree. The reasons for this hierarchical structure will be discussed later. The file, however, can be viewed in other ways different from the sequential tree structure. For instance, associational trails can be drawn throughout the file and followed. Thus the AHI file is capable of modeling Bush's .FTN=1; 1. V. 1bush, "As We May Think", Atlantic Monthly, July '45. .FTN=0; network of associational trails as well as a sequential hierarchical text.

2a1 1.1 %THREE %TYPES %OF %BASIC %ENTITIES

2a1a A. Statements

2a1b B. Vectors

2a1c C. Keywords

2a1c1 These three types of entities are stored in statement data blocks (SDB's), vector blocks, and keyword blocks, respectively. In addition, the hierarchical structure of the text is stored in ring blocks. We will only discuss the statement data blocks and ring blocks and their relation to the main file; the access and storage of vectors and keywords is very similar and so do not need to be discussed separately. (In the present version of the system, the only types of vectors that can be stored are straight lines, and no sketching facility exists other than defining the straight line by its endpoints. There is no rubberbanding. A sketching facility is planned for a future version.)

2a2 1.2 %STATUS %TABLES

2a2a Associated .FTN=1; handwritten note: All status tables are in the %file %header (block 0) .FTN=0; with each set of blocks (there are four sets of blocks: the ring blocks, the statement data blocks, the vector blocks, and the keyword blocks9 there is a small status table which has an entry for each block of its kind. Thus there is a ring status table, a statement status table, a vector status table, and a keyword status table. The entry for each block in the status table simply points to a "global" random file status table block, which gives the location of each block, whether in core or on drum. (See Fig. 1)

2a2a1 .ILL=1; FIG. 1 Status Tables

2a3 1.3 %RANDOM %FILE %STATUS %TABLE %BLOCK

2a3a The random file status table block is a block that contains an entry for every block of every type in the system (actually, there is an RFSTB for each (active file). Each entry tells whether the block is in core or not, or whether it is unallocated (i.e., not being used at the present time and can be allocated when a file is exa, expanded through editing). The entry also gives the information on where the block is located, on the drum or in core. It is through the RFSTB that each desired block is located by the system:

2a3b As we saw in 1.2, each ring block is mapped into an entry in the ring status table, each statement data block is mapped into an entry in the statement status table, etc. Then each entry in each status table points to an entry in the bigger RFSTB (at present there are a maximum of 64 blocks in the RFSTB), one containing pointers to the actual location of each different block in the file. This double-table method of location of each block is to facilitate control of the allocated and unallocated area on drum, and for garbage collection; furthermore, this central location mechanism allows blocks to be moved in the system, without internal pointers having to be modified.

2a3c The RFSTB contains information other than just a pointer to the block, whether in core or on drum. One area of the RFSTB if it is less than zero, indicates the block is on the drum. If the number in this area is negative, it is the number of free words in that block. This is to prevent needless retrieval of the block for additions if there is not enough room on it for the desired update. If this area is greater than zero, the block is in core and the number is the core address of the block. There is another area in which an indicator (at present a-2) says free space is too small to consider going there. This is computed from the average lengths of the statements.

2a3d The ring status table at present has eight entries, one for each of eight ring blocks. This number is expandable, and is an assembly time variable. There are probably twice or three times as many SDB'S .FTN=1; handwritten note: about 55 presently .FTN=0; and therefore the statement statusstable is correspondingly bigger.

2a4 1.4 %STRUCTURE %OF %THE %RING %BLOCK

2a4a Each statement is represented in the data structure both by its associated text (see SDB's) and by a ring element, that is, by an element of a ring that contains the hierarchical tree structure of the file and points to the text associated with each node (statement9 in the tree. The ring is broken into ring blocks, each of which is 1024 words long. Each ring block has a header and then is composed of ring elements, each four words long, one ring element per statement (See Fig. 2):

2a4a1 note: pointers to ring elements are called PSTD'S (permanent statement ID) and never changes during the existence of a statement.

2a4a2 .ILL=1; FIG. 2 Ring Block Elements .ILL=0;

2a4a2a Pointer (PSDB): the internal pointer to the statement text in a statement data block (SDB). (The structure of an internal pointer (symbolized by P) will be discussed in Section 1.6)

2a4a2b Flags: (this would require updating the entire ring after every structure change).

2a4a2b1 1) on if the statement has a name

2a4a2b2 2) pattern filter tested?

2a4a2b3 3) pattern filter result?

2a4a2c Successor: a pointer to the ring element (anywhere in the ring) of the next succeeding statement on the same level.

2a4a2d Sub: a pointer to the ring element of the first statement in the level directly below the current statement.

2a4a2d1 The sub and successor pointers define the hierarchical tree structure.

2a4a2d2 SUB points to this ring element if there is no substructure

2a4a2e H: on if this statement is a head

2a4a2f T: the last successor on each level points to the source (up 1 level) of that level, thus providing a back pointer. The T bit is set when it is the last successor.

2a4a2g Name hash: this is a 24-bit hash of the (optional) statement name. Thus when jumping by name, we need only scan each ring element sequentially to get correct statement. This may be done sequentially since each file generally consists of only about 300 statements.

2a4a2g1 note: unused ring elements in a given ring block are linked together on a free list -- pointed to by a pointer in the block header.

2a5 1.5 %STATEMENT %DATA %BLOCK (%SDB)

2a5a The statement data blocks (SDB's) are simply areas in which to store the statement text. No structure or hierarchy is part of the ;sdb's since that is taken care of by the ring blocks. The system tries to put all sequential statements in the same block to save on drum I/O. The process of initial generation and placement of statements in the SDB'S will be discussed in section 2. (See Fig. 3)

2a5b .ILL=1; FIG. 3 Statement Data Block Elements .ILL=0;

2a5b1 CKSUM: this is a checksum to check against hardware I/O errors in reading the statement data block from drum. Before writing out on drum, the system adds up all words in the SDB and stores the sum in CKSUM. On read-in, it re-adds the words and checks to see if the sum is the same. .FTN=1; Handwritten note: All 1k file blocks are checksummed in this way .FTN=0;

2a5b2 Header: this contains the initials, date, and time of last user and change, fields which can be used as a later means of retrieval.

2a5b3 PSID: Back pointer to ring: this is an internal pointer (of type p) to the ring element representing this statement in the text hierarchy, i.e., the ring element which points to this statement.

2a5b4 Flags: the first bit indicates whether this SDB element is garbage, and is used when compacting the SDB Other bits indicate whether it is difficult to format the statement on the display, that is, if the statement contains things like underlining or flicker. If it is difficult to format, the low speed scanning/ formatting routine is used. Otherwise the high-speed routine is used. This saves up to 50% on time. .FTN=1; handwritten note:

(this is not actually done on current system, but just wait) .ftn=0; (Other bits for other things.)

2a5b5 Text: the text is stored in the statement data block as follows: there are two kinds of characters, (1) 8 bit character, with the high order bit off, and (2) 16 bit characters. If the high order bit is on, this signals that the character is a 16 bit character. The seven next high bits signify font, etc., of the character represented in the second eight bits. The different qualities of each character are underline, blinking, italics, boldface, etc. The user can make up his own special characters and the system will insert it. This is done by giving the special character a number. It takes less than 10 msec. to reformat a display. (I/O not included)

2a6 1.6 %LOCATING %STATEMENT %IN %THE %DATA %STRUCTURE

2a6a 1.6.1 %Mapping %Statements %to %Ring %Block %Elements %Through %the %Internal %Name (%Pointer) %P

2a6a1 When statements are created, they are assigned by the freelist allocator to open positions in a ring block (to a ring element) and assigned to statement data blocks according to the "garbage bits"; they are also assigned an internal (position related) name in the ring block denoted by p. All ring block vacancies are kept on the freelist. The internal name P in the ring block is thus assigned by getting it off the freelist (creating a map from statement name position to internal name, and from internal name to block position) as described below:

2a6a2 Say that we want to retrieve statement P, an internal name (pointer) of the type found in the successor and sub fields of the ring element. It is listed in the file header where it gives the point in the ring where the file starts, i.e., it points to the ring element representing the first statement in the file.

2a6a3 To get statement P (11 bits) we look at P*4, which is 13 bits long (See Fig. 4). The upeer 3 bits are an index on the 8-entry ring status table (RST). The entry in the RST points to an entry in the random file status table block (RFSTB). This entry in the RFSTB tells us whether the ring block contains the containing the desired ring element is in core or not, or whether it is unallocated (in which case an error condition exists). The ring block is brought into core if necessary. The lower 11 bits of P*4 then form an index relative to the start of the ring block that bring us to the appropriate ring element. Thus from the internal name of the statement we retrieve the desired ring element.

2a6a3a .ILL=1; FIG. 4 Structure and Mapping of

Internal Pointer P

2a6a4 Notes on the file header: this contains pointers to all status tables and their lengths, and information on the virtual memory map. It also contains bibliographic information which may be used as a means of retrieval: last time written into, username, initials of last user, jump delimiters (these are the marks that delineate a jump, and in the present veesion are general parentheses), average length of statement (determined by how much activity over periods of time). This information is all contained in the first 1k words. An interesting feature is that the ;ts system will accept any amount extendable to 1K without using excess drum space. .FTN=1; handwritten note: 256 word chunks, I believe .FTN=0;

2a6a5 The first ring element at the start is dummy; and is the start of the file. When the system rewrites the file on drum after use, it searches to the first semicolon and puts in place of what is ttere the file description: username, initials, date

.FTN=1; handwritten note: filename .FTN=0; initials, date and time, etc. of last use.

2a6b 1.6.2 %Mapping %From %the %Ring %Element %to %the %Text %of %the %Statement %(See %Fig. %5)

2a6b1 .ILL=1; FIG. 5 Mapping from Ring Element to Text .ILL=0;

2a6b2 Now that we have the appropriate ring element relating to statement P (see Fig. 2) for the structure of ring element, we can retrieve the statement-text for filter/format/display. The system takes the "pointer to text" in the first word of the ring element. This pointer is of the same structure with respect to the statement in the statement data block as P*4 is to the ring element of the ring block. Thus the high bits are an index on the statement status table. The entry in the SST points to an entry in the RFSTB, which in turn points to the location of the appropriate SDB in which the desired statement is located. Once we have the appropriate SDB, the 10 low order bits of the original pointer point to the desired statement, relative to the start of the SDB (See Fig. 3)

2a6c 1.6.3 %Generating %a %Sequence %of %Statements

2a6c1 Given the appropriate individual statement P (see Fig. 3 for structure of the statement entity), the sequence generator now takes the statement text for filtering/formating/display, as described in Section 3.

2a6c2 Which statement is taken next depends on the sequence being followed by the sequence generator. If the sequence generator is following the basic hierarchical tree structure, it will look at the ptr-to-sub field in the ring element (Fig. 2), and use that pointer as it used P above. (However, if a filter is set for a specific level and statement P was on that level, the sequence generator will ignore the sub field and take the ptr-to-successor field. .FTN=1; handwritten note: has to find the level of the first statement and keep track of it .FTN=0;

2a6c3 The sequence generator, however, may be following an associational trail. If this is the case, the content analyzer will scan the statement-text P for the appropriate trail marker. If it finds the appropriate trail marker in the statement-text, it will hash the name in the trail marker, and scan the name hashes of the ring elements until it finds the correct ring element, and continue generating statements from there. If the appropriate trail marker is not found, it will follow the tree structure as able. .FTN=1; handwritten note: (by hierarchy until a trail marker is found) .FTN=0;

3 2. %DATA %STRUCTURE %MODIFICATION

3a The data structure is modified through the basic editing commands (delete, insert, replace, move, copy, break/join) which are described briefly below in Section 4. System features and facilities are described more completely in the "NLS User's Guide" (an SRI publication).

3b We will describe how the data structure is modified for an insert; the other types of edit-modifications are all similar. If the edit is an insert, it is an insert in the middle of a statement. By system definition, all editing is based on the statement. The user types in the appropriate insert command, hits the point of insert with the mouse, and types in the insert. The insert typed appears on the screen in the literal type-in area. If the user decides the insert is complete, he hits the command-accept button. The system then makes the modification of the data structure as follows:

3c The system computes the new length of the statement by taking the old length of the statement in the statement data block and adding the length of the insert. The system then finds a free area on one of the statement data blocks of sufficient length to put the new statement. It tries to put the updated statement on the same SDB. If the edited statement does not fit in that SDB the system tries to compact the block. If that would give enough space, the system goes to the previous ring element and sees what SDB that statement is stored on and tries to fit the newly updated statement on that block. If it doesn't fit there, the system looks through the ;sdbst to find any free area and fits it in anyplace.

3d Now that the appropriate space is allocated, the updated statement is constructed. This is done by copying the header of the original statement and the text up to the insert point, adding to this the literal type-in, and copying the rest of the text of the statement. Then the "ptr-to-text" in the associated ring element is changed to point to the new statement, and the garbage bit is set in the original statement.

3e When any statement is edited, the system checks to see if there is a statement name, or label. If there is, it is rehashed and replaced in the ring element. Thus labels are always updated.

4 3. %REDUCING %THE %DATA %STRUCTURE %TO %A %SCREEN %DISPLAY

4a The process of scanning the data structure to retrieve and display the desired text has four basic parts: (1) the sequence generator (as discussed briefly in Section 1.6.3), (2) filtering, (3) formatting, and (4) display.

4a1 3.1 %SEQUENCE %GENERATOR

4a1a The sequence generator is the routine that actually scans the data structure and generates the sequential text. Basically it generates a list of statements. There are three types of sequences that can be generated:

4a1a1 3.1.1 %Tree

4a1a1a This is the default hierarchical structure that is generated and is simply the sequential text of the main associational trail of the text, ordered in a hierarchy of statements.

4a1a2 3.1.2 %Trails

4a1a2a The trail feature is used to set up statements in such a way that only a particular set of statements will be displayed and in a particular order. The set of statements is called a trail, and is an associational trail that criss-crosses the default (main) trail; it provides a manner other than the normal sequence in which to read the text. A trail marker is set up for a particular trail of statements; the pattern for this marker can be a complex syntactical form and is followed by the content analyzer (described in an SRI publication).

4a1a2b The trail feature is used to set up statements in such a way that only a particular set of statements will be displayed and in a particular order. The set of statements is called a trail, and is an associational trail that criss-crosses the default (main) trail; it provides a manner other than the normal sequence in which to read the text. A trail marker is set up for a particular trail of statements; the pattern for this marker can be a complex syntactical form and is followed by the content analyzer (described in an SRI publication).

4a1a2c Trail markers are thus used to mark turning points from the normal sequence of statements, as a sign post to the next statement in the trail. Each time a marker appears in a statement it is followed by a statement name in parentheses that is the name of the next statement. Between trail markers statements are displayed in normal sequence. The trails can be followed only in the forward direction; there is no capability for inverting the trail when moving backwards through the text. (SRI claims that with the complex content-analyzer, this is unnecessary.)

4a1a3 3.1.3 %Keywords

4a1a3a The keyword system permits a user to construct a specially formatted catalog file containing references to other files and capable of being reordered automatically according to some chosen set of weighted keywords. When reordered, the file lists references in order of relevance, according to the choice and weighting of keywords.

4a1a3b The keywords are attached to a statement. The system keeps a list of the keywords containing for each keyword a short description of the keyword, and the labels of statements tagged with this keyword. This list is visible to the user and can be changed by him. The system also keeps a list of the file-reference entries, that is, a file of any statement name tagged with a keyword, and a list of the keywords it is tagged with following it. Thus one keyword can be attached to any number of statements, and one statement may have any number of keywords attached to it.

4a1a3c The keyword system is used mainly as a retrieval-by-keyword system. The user selects desired keywords and weights them according to importance. A negative weight can also be used to blackball any keywords. According to SRI the weights on the keywords allow more flexibility than straight Boolean retrieval functions on keywords; after the user has selected keywords and weights, the system goes to the list of keywords and picks out all statements tagged with the selected keywords. For each statement selected the system computes the weights of keywords attached to it, and displays the names of the statements in order of highest total weight. Statements with a negative total weight are not displayed. The user may then access the referenced files by using the jump command on the statement names.

4a2 3.2 %FILTERING

4a2a Facilities included are: level specification, branch only only, subfile, content analyzer, trail flags, Iterals, search for trail flags and literal text, etc.

4a2b After the main structure is generated and filtered, it is formatted.

4a3 3.3 %FORMATTING

4a3a The formatting sets the following, and other, variables of display:

4a3b Statements numbers4

4a3b1 Statements numbers: the number of lines of each statement to be displayed is variable; headers, time/initials/labels can be on/off.

4a3c View change: character size, page size and dimensions, etc.

4a4 3.4 %DISPLAY

4a4a After the statements have been filtered out, they are displayed. The main display of the generated/filtered/formatted structure is in the file area of the screen. There are a number of one-dimensional registers used for man/machine interaction:

4a4a1 1. Echo register. This displays the last six characters typed by the user, for feed feedback.

4a4a2 2. Command display line. This is a line which says what command is in the process of being executed.

4a4a3 3. Name register. nc Displays user's name (this is on a multi-terminal system).

4a4a4 4. View specification areas. There are three view spec areas, and these are set according to the formatting variables described in Sec. 4.28.

4a4a5 5. Message area. An area for system messages to the user, such as error messages.

4a4a6 6. Literal type-in area. When the user is typing in an insert or delineating a command, the characters typed are displayed in this area.

4a4b There is, in addition to the file area, another two-dimensional area, the freeze area. This freeze area is used to "freeze" statements designated by the user so that they remain unchanged above the file area, with the file being then displayed in the file area. The freezed statements remain unchanged despite any text manipulations or file searching that goes on in the file area. (In a future version, the freeze area will be done away with, and instead the user will be able to multi-window any number of windows. Each window will be a full file area, with all one-dimensional registers in each window. They can be any shape or size any place on the screen. With multistations, a window can be assigned to a station, giving the users at two different terminals the ability to decide who holds the chalk and who holds the eraser in each window.

5 4. %SYSTEM %FEATURES %AND %FACILITIES

5a 4.1 %EDITING

5a1 The basic editing commands are delete, insert, replace, move, copy, set, and break/join. All are self explanatory, except set and break/join. The set commands allow the user to change the font on any text string. The fonts are: capital, lower case, italic, roman, boldface, no boldface, flickering, non-flickering, underline, no underline. The break and join commands allow the user to break a statement into two statements. statements; the join command adds a text string onto another statement. The break and join commands are the only commands that operate across statement boundaries. All the other editing commands are specialized: for example, the insert commands are insert character, insert word, insert text, insert invisible, insert statement, insert branch. The specialized commands make it easier for the system to make the edits; the rationale for specialization is that since you have to type the command in you may as well specialize, and economies in data structure manipulation may be achieved (e.g., moving an entire branch of the tree).

5b 4.2 %OTHER %FEATURES

5b1 4.2.1 %Invisibles

5b1a When editing, invisibles such as spaces and tabs can be displayed by marks, and thus can be deleted.

5b2 4.2.2 %Labels

5b2a Labels are statement names and are used for retrieval purposes by jumps, links, and keywords. They are inserted as part of the text, that is, with an insert command. A label is simply a variable-length character string that appears %at %the %beginning %of %a %statement %in %parentheses. These labels can be changed or deleted as if they were regular text.

5b2b Duplicate labels can be created. A jump to a label results in a jump to the first occurrence of that label, since the system sequentially scans the name-hash field of the ring elements. A feature contemplated for incorporation in the system is a "look for next occurrence of this label" jump to resolve duplicate labels.

5b3 4.2.3 %Links

5b3a A link is an association to another statement, i.e., it is a jump to another statement that can be taken at the option of the user. The link can be in the current file or in another file. There are four parameters to a link: three (the user name, filename, and label) define the point linked to. The fourth is the view specifications on the text linked to. This is an interesting feature: that view specifications can be changed on all links.

5b3b The link structure is a regular text string inserted in the text as if part of it, and is in parentheses in a syntactic form. Like labels, the link is just regular text until it is used. It can be edited at any time. When the user decides to take a link, he hits a character with the bug. The system scans forward with the content-analyzer until it picks up the nearest link structure in that statement, and jumps to the label. The link is taken by use of a jump command.

5b4 4.2.4 %Intrafile %Return %Ring

5b4a Whenever any jump is made within the file, a new entry is made in a list called the intrafile ring. Each of these entries gives a display start and a set of viewspecs. 1a pointer indicates the current view on the list. Each time a jump is executed, the new information is written ahead of the pointer and the pointer is moved forward. On a jump return or jump ahead, the pointer is simply moved backward or forward and no new entries are made or any deleted. The list holds a maximum of six entries, and is circular.

5b5 4.2.5 Interfile Return Stack

5b5a This works much like the intrafile stack except that it is concerned with jumps between files. The differences with the intrafile stack are: (1) the length of the list is variable, and depends on the amount of information in the links used, (2) the list is not circular, a new entry is made on the stack whenever any interfile jump is taken or whenever a new file is loaded with a load file command. (See section on multifiles for more details.)

5b5b There are no backpointers from a link, the same as with trail markers. Thus if a label that is linked to is deleted, there is no user notification that a link has been made inoperable. Also, since link structures are entered as simple text, the label in a link structure does not necessarily exist. A link or jump to a non-existent label results in an error condition.

5b6 4.2.6 %Jump

5b6a The jump command brings the desired statement to the top of the display.

5b6b There are four basic types of jumps: (1) jumps to a specified label name, (2) jumps to links, (3) jumps through the tree structure, and (4) jumps among different files.

5b6b1 In case 1, the label or statement name to be jumped to can be specified by either a word-selection via the mouse or a literal entry from the keyboard.

5b6b2 In case 2, the statement defined by the specified link is placed at the top of the display. More detail is given in section 4.2.3.

5b6b3 The case 3 commands allow jumps to the next substatement, the next successor, the statement of which the selected statement is a substatement, the previous statement, the head of the file, the end of the file, the end of branches, and many other links on the basis of tree and file structure. For more details see the "NLS User's Guide."

5b6b4 The case 4 commands allow the user to load a number of files into the system and to jump freely among them. These will be discussed in Section 4.2.9.

5b6b5 There is one other type of jump, the jump-ahead/return. Whenever any type of jump within the current file is executed, the system keeps track of it, and a ring is maintained keeping a sequential track of all views that have been used. These commands allow the user to return to a previous view or to move forward after a jump return to the latest view. (See Section 4.2.3 on links for a description of this intrafile ring.)

5b6b6 A special feature of jumps is that almost all jumps allow the user to change the view specifications of the area jumped to from those of the current text. In addition, each jump saves the viewspecs of the area jumped from in the intrafile ring, so that on a jump return the text is viewed as before.

5b7 4.2.7 %Pointers

5b7a Pointers make it possible to select entities that are not on the display. The entity has a pointer fixed on it while it is on the screen of not more than three characters. To select the entity at any time, a mouse button is depressed and the name of the pointer is entered from the keyboard. This is exactly equivalent to making a direct bug selection of the character that has the pointer on it.

5b7b The list of pointers can be displayed and one may use it to jump to the individual pointers.

5b8 4.2.8 %View %Specifications

5b8a The view specifications (viewspecs) are parameters that control the way in which statements are displayed. The parameters are: indenting on/off; names on/off; display file as tree/normal text; keyword reordering on/off; display of statement signatures on/off; branch-only on/off; content analyzer on/off; trail feature on/off; pointer display on/off; number of lines displayed; number of levels of statements displayed and a few others. These canbe set in three ways: with the view set command, from the special keyset, or during certain commands such as jump.

5b8b These parameters are always displayed in the upper left corner of the screen with a single letter denoting each. When they are capable of being changed by the user, they are displayed with larger letters.

5b8c There is a relative level control, which allows changes to the level parameter set by the user to be interpreted relative to the level of the first statement in the display. The user can also change other viewing parameters. These include the type of mark the cursor leaves, the number of characters in a line of text, the number of spaces indented for each level, the number of lines in the text area, the spacing between lines, size of characters, etc.

5b9 4.2.9 %Multi-%files

5b9a When a file is loaded or jumped to, it is "opened" and displayed; no copy is created, rather the file is viewed directly from the disk. For reasons of file protection, if any changes are made, it becomes impossible to continue direct viewing, so the system creates a working copy when an edit is made. In fact, this working copy is not created until all core is filled and not necessarily on the first edit. In this way the system does not make a working copy until it definitely has to. When the system creates the working copy it copies the displayed file to it, closes the displayed file, and from then on all work is done in the working copy. No working copy is created when the user is just browing. Olthis browsing. This is done since most users just look at files and do no editing.

5b9b Files are loaded by the load command or by an interfile jump command. Entries are made in the interfile stack as files are loaded (see lsection 4.2.5). The working copy and the checkpoint file are never entered in the stack.

5b9c One feature of the multi-files is that the user can create a checkpoint file at any time. This writes the present working copy out on the drum under the name checkpoint.

5b9d The interfile stack can be used like the intrafile stack to go back and forth among views on different files. Only one working copy at a time can be created, and can be looked at any time, even if a file other than the one of which a working copy was made has been currently loaded.

5b10 4.2.10 %Freeze

5b10a The freeze feature freezes a single statement with the preesent view. The frozen statement will appear at the top of the screen whenever frozen statements are being shown, with the main text display on the under part of the screen. A fixed number of statements can be frozen, and are displayed in the freeze area in the chronological order frozen.

5b11 4.2.11 %Tree-%Display %Feature

5b11a This allows the user to see the file as a tree structure, or in the hierarchy form, instead of normal text. The tree structure shows the relationships of statements in the file to each other. This is done by indenting the differing levels of the tree to different depths, much like an outline form. This can be turned on or off by the view specifications.

5b12 4.2.12 %Statement %Numbering

5b12a The system numbers each statement Dewey Decimal fashion according to the tree structure. This numbering is computed at display time. The numbering can be turned off by the view specifications.

5b13 4.2.13 %Vectors

5b13a The vector package allows the user to create simple line drawings, with labels for jumps. The vector is drawn by specifying the endpoints with the mouse. Either endpoint of a line can be translated, and the entire drawing and any label can be translated. These vector labels can be used as jumps to that statement name.

6 5. %FUTURE %FEATURES

6a 5.1 %MULTIWINDOWS

6a1 This may have been inspired by our multiwindows. Theirs, however, is fancier in conception. This would allow any size and shape windows to be defined, and each window to be a self-contained viewing area with all the parameters as described for the single screen display. Their multiwindow facility could also assign different windows to ifferent users. This assignment is done by the time sharing system, though; the only programming problem is the protocol: who holds the eraser in each window.

6b 5.2 %VARIABLE %SYMBOLS

6b1 This would allow the user to define a variable symbol for text, links, etc. The symbol would be filled in with text at display time, like an assembly time variable. Alternately, the variable symbol could simply be permanently defined at a later time.

6c 5.3 %WEIERSTRASS %ALGORITHM

6c1 Currently the system uses a display map technique for detecting bug hits. A future plan is to use the Weierstrass Algorithm of continually subdividing the screen to find the line closest to the bug mark, which would be the line hit.

6c2 The hierarchical structure allows the text to be set out in a tree form very easily. The question of advantage of this over traditional text was discussed with lengelbart. He said that the hierarchical statement-oriented structure was selected just as a starting point and empirically has proven to be more helpful to users in terms of visualizing the text. lhe insisted there is no premeditated reason toward this structure, nor need it be imposed on the user.

6c3 The statement oriented quality limits the flexibility of editing somewhat. From our point of view, there is no editing across statement boundaries, for instance. Jeff said that this limitation is of no real importance since as users gain familiarity with the statement oriented system, they learn to make statements complete thoughts, and so editing across statement boundaries is not really necessary; the limitation is only on traditional thinking with traditional text. This is the same reason Engelbart stated for using hierarchy: the user quickly adapts to the structure provided him.

6c4 One advantage of the statement oriented structure is that to move a branch or a statement requires no actual movement of text, but just the changing of a few pointer.

6c5 There is great effort not to let the user hurt himself when he cannot see the entire tree structure due to filters. For example, a user cannot delete an entire statement. There might be substatements below that are filtered out that he might inadvertently delete: he must give a delete-branch command and delete the entire branch.

Description of Data Areas of NLS

1 Data forms and terms

1a (A-string) An A-string is an array of words which contains an ASCII character string in the following format: The first word contains an integer that is the maximum number of characters that the string can contain. The second word is an integer that is the current length of the string. The string starts in the next word and is packed three characters per word. A null string is indicated by a current length of -1, a one character string by zero, etc. A-strings are usually handled via routines APCHR, APSR, LDCHR and others in the UTILTY overlay.

1b (General Stack) The general stack (array STACKD) is used by the SPL routines for return locations and arguments, and by the Content Analyzer pops in SDBMNP. The stack pointer is STACK, which points to the current top word (being used) in the stack.

1c (PSID) PSID stands for Permament Statement IDentifier. It is a 11 bit integer between zero and 2047 which NLS uses as identification for a statement. It remains unchanged as long as that statement is in the file. That is, the PSID for a statement is not changed, even though the text may be completely replaced or the statement moved in the file structure. When the statement is deleted, that same PSID may later be used to identify a different statement.

1d (RING element) A PSID refers to a statement. Every statement has a RING element, and the PSID can be used to find the ring element. The RING element contains various fields, and these are usually read via the GET routines in the UTILTY overlay. A RING element is a four word block that contains the following:

1d1 First word: The last five octal digits (15 bits) contain a PSDB. The first three octal digits contain flags which mean:

1d1a Bit 0: Statement Name is present or not

1d1b Bit 1: This statement has been tested against current pattern

1d1c Bit 2: On if this statement passed the pattern test

1d1d Others: unused at present

1d2 Second word: bits 0 to 10 contain the PSID of the SUB statement. If there are no SUB statements, the PSID of this statement is in this field. Bits 13 to 23 contain the PSID of the SUC (successor) for this statement. Bit 11 is on if this statement is a HEAD (first in a plex). Bit 12 is on if this statement is a TAIL (last in plex). If it is a TAIL, the SUC is the PSID of the SOURCE statement, and there is no successor.

1d3 Third word: The hash code for the name is here. It is zero if there is no name.

1d4 Fourth word: This contains the PVDB if there is a vector picture with this statement, otherwise it is zero.

1e (work area) Several routines require work areas as calling arguments. A work area is simply an array of cells in a read-write page. The address of the work area is usually provided in the X, and occationally the work area needs to be initialized in a certain way.

1f (SDB) An SDB is a STatement data block, a variable length block of words in a random file text block (type 1). SDB's have a seven word header that indicates:

1f1 (1) length of stb (2) psid (3) date (4) time (5) initials (6)number of characters (7) characters in name

1g (VDB) A VDB is a vector data block, a variable length block of words in a random file vector block (type 2). VDB's have an eight word header that indicates:

1g1 (1) length (2) psid (3) date (4) time (5) initials (6) # of lines (7) height (8) spacing

1g2 Height is given in raster units

1g3 The last word contains the spacing flag which is set if space is to be left for the drawing when the display is created

1h (PSDB) A PSDB is a 15 bit pointer to a statement data block (SDB). The PSDB in a ring element indicates where the text for that statement is.

1i (block type) Every random file block has a type number in it. The type number indicates that the block is:

1i1 (0) Not legal

1i2 (1) A text block (a block containing SDB's)

1i3 (2) A vector block of VDB's

1i4 (3) A ring block

1i5 (4) The query block

1j (block number) The random file blocks are numbered sequencially. Block zero is the header block. The block number is an index into the random file block status table.

1k (block status table) Their are two levels of status blocks for providing information about the random file blocks. The random file block status table (RFBS) indicates whether each block is in core or not, and if not, how many words are free in that block. For each type of block, there is a block status table with one entry for each block of that type. An entry indicates which random file block status entry corresponds to that block.

1l (overlay) An NLS overlay is a one page (2K) block of code, usualy reentrant. OVerlays are usually not read from an IO device when needed, but are "around" and ready to run when NLS is started. Excatly whether the overlay is in core or not is up to the time sharing system.

1m (relabeling)

1n (overlay number)

1o (address)

1p (overlay address)

1q (interrupt)

1r (hash code)

1s (buffer)

1t (freezing)

1u (thawing)

1v (literal)

1w (statement vector)

1x (statement number)

1y (T-pointer)

1z (T-string)

1a@ (state)

1aa (subroutine)

1ab (argument)

1ac (pointer)

1ad (register)