NDOT

```
< NLS, NDDT.NLS;7, >, 1-NOV-76 13:25 JDH ;;;;< NLS, NDDT.NLS;55, >,
26-JUN-74 11:09 KEV ;
FILE nddt  % L10 <rel-NLS>nddt.rel; %% (L10,) (rel-nls,nddt.rel;,) %
UND-OK
% pattern for getting table of contents - $NP "( $LD ") ;  %
%* TABLE OF CONTENTS OF CONTAINED ROUTINES %
%Defines and declaractions%
   %Defines...alphabetically listed (not compiled)
      DEFINE addop = 1#;
      DEFINE addressmode = rd.adrmod#;
      DEFINE areg1 =db[12]#;
      DEFINE areg2 =db[13]#;
      DEFINE areg3 =db[14]#;
      DEFINE areg4 =db[15]#;
      DEFINE backupchar = bkjfnjsys()#;
      DEFINE bptblentsz = 4#;
      DEFINE bptblsz =40#;
      DEFINE calldefineproc = &calprc_defineproc; calprc#;
      DEFINE callmother=&calprc_mother; calprc#;
      DEFINE callreplaceproc = &calprc_rd.replact;calprc#;
      DEFINE calltyperoutine = &calprc_typeroutine; calprc#;
      DEFINE came =312B9#;
      DEFINE camg =317B9#;
      DEFINE camge=315B9#;
      DEFINE caml =311B9#;
      DEFINE camle=313B9#;
      DEFINE camn =316B9#;
      DEFINE continsig = 2#;
      DEFINE currentfld=&rd+rdsize+(IF fieldnumber >= numberfields THEN
      err($"Field Out of bounds") ELSE fieldnumber)*fielddescsz#;
      DEFINE currenttypemode =db[21]#;
      DEFINE daughter = rd.kidrec#;
      DEFINE ddtlimit = 500B#;
      DEFINE ddtsymbol = symtype.ddtsm#;
      DEFINE ddttablesz = 38#;
      DEFINE defaultmode = 0#;
      DEFINE defaultreplace = 1#;
      DEFINE defineproc=rd.rdef#;
      DEFINE divop = 4#;
      DEFINE dynamicrecord = rd.rdproc#;
      DEFINE enterddt =!JSP 0,770002B#;
      DEFINE entity = rd.enttyp#;
      DEFINE escchar = db[17]#;
      DEFINE fielddescsz=2#;
      DEFINE fieldentity = entity=fieldtype#;
      DEFINE fieldnumber = rd.rindex#;
      DEFINE fieldptr=fld.fpointr#;
      DEFINE fieldstart = &rd+rdsize#;
      DEFINE fieldsymbol = symtype.fldsig#;
      DEFINE fieldtype=2#;
      DEFINE fixoverflow =fixoutofbounds(&rd, overflowaction,
      overflow)#;
      DEFINE fixunderflow =fixoutofbounds(&rd, underflowaction,
      underflow)#;
      DEFINE framebase=db[24]#;
      DEFINE framebasetype =2#;
```

```
DEFINE framemark =[framep].marklocfield#;                          053
DEFINE framep = db[19]#;                                           054
DEFINE framesize=db[23]#;                                          055
DEFINE frametop =[stacktop].marklocfield#;                         056
DEFINE frametoptype =1#;                                           057
DEFINE global = 1#;                                                058
DEFINE gosig = 1#;                                                 059
DEFINE indirectmode=1B6#;                                          060
DEFINE indirectsym = indddtsym#;                                   061
DEFINE initialised =rd.rdinit#;                                    062
DEFINE internalfield = internalsym + 1B9#;                         063
DEFINE internalstacksym = internalsym+1B6#;                        064
DEFINE internalsym = 2B6#;                                         065
DEFINE jrst =254B9#;                                               066
DEFINE jsp = 265B9#;                                               067
DEFINE lbptbl=db[26]#;                                             068
DEFINE lbptblentsz = 1#;                                           069
DEFINE lbptblsz=nbkpts#;                                           070
DEFINE lc =db[36]#;                                                071
DEFINE leftbyteptrrj = 4307B8#;                                    072
DEFINE lv = db[16]#;                                               073
DEFINE macro = 0#;                                                 074
DEFINE macromode = addressmode=macro#;                             075
DEFINE maxnumfields = 50#;                                         076
DEFINE maxprms = 20#;                                              077
DEFINE maxwordsinstring = 402#;                                    078
DEFINE micro = 1#;                                                 079
DEFINE micromode =addressmode=micro#;                              080
DEFINE mother = rd.momrec#;                                        081
DEFINE movei =201B#;                                               082
DEFINE mulop = 3#;                                                 083
DEFINE nbkpts=10#;                                                 084
DEFINE nextchar = binjsys()#;                                      085
DEFINE noindexmask = 777760777777B#;                               086
DEFINE nparms=db[22]#;                                             087
DEFINE number = symtype.numsym#;                                   088
DEFINE numberfields = rd.nfield#;                                  089
DEFINE numericmode = 1#;                                           090
DEFINE opcall0=1#;                                                 091
DEFINE opcall1=2#;                                                 092
DEFINE opcallm=3#;                                                 093
DEFINE opcodmask =9M9#;                                            094
DEFINE optabl = 24B#;                                              095
DEFINE overflow=1#;                                                096
DEFINE overflowaction = rd.topact#;                                097
DEFINE proceedaction = 1#;                                         098
DEFINE proctblentsz =2#;                                           099
DEFINE proctblsz = 10#;                                           0100
DEFINE progm =db[9]#;                                             0101
DEFINE progp =db[7]#;                                             0102
DEFINE programsymbol = symtype.progsym#;                          0103
DEFINE progrp =db[10]#;                                           0104
DEFINE progs =db[8]#;                                             0105
DEFINE quitsig = 3#;                                              0106
DEFINE radix=8#;                                                  0107
DEFINE rdsize = 6#; %%Number of PDP10 words in record%%           0108
```

```
DEFINE recordentity = entity=recordtype#;                                0109
DEFINE recordname = rd.rdef#;                                            0110
DEFINE recordsize = rd.rsize#;                                           0111
DEFINE recordstart = rd.raddr#;                                          0112
DEFINE recordtype=3#;                                                    0113
DEFINE recp =db[37]#;                                                    0114
DEFINE recprecordaddr =2288#;                                            0115
DEFINE recprecsize =2222B8#;                                             0116
DEFINE reg1 =db#;                                                        0117
DEFINE reg2=db[1]#;                                                      0118
DEFINE reg3 =db[2]#;                                                     0119
DEFINE reg4 =db[3]#;                                                     0120
DEFINE reg5 =db[4]#;                                                     0121
DEFINE reg6 =db[5]#;                                                     0122
DEFINE reg7 =db[6]#;                                                     0123
DEFINE replaceaction = rd.replact#;                                      0124
DEFINE returnloc =[framep].retlocfield#;                                 0125
DEFINE rfnmfg = db[20]#;                                                 0126
DEFINE ribsize = 106#; %%rdsize + maxnumfields*fielddescsz%%             0127
DEFINE rpreg =db[11]#;                                                   0128
DEFINE seqsymbol = 4B6#;                                                 0129
DEFINE sequencename = symtype.seqsym#;                                   0130
DEFINE setfieldptr =&fld _ currentfld#;                                  0131
DEFINE specialsymbol = stacksymbol .V sequencename#;                     0132
DEFINE stackref=sequence#;                                               0133
DEFINE stacksymbol = symtype.stksym#;                                    0134
DEFINE stacksymv =1B6#;                                                  0135
DEFINE stacktop=db[25]#;                                                 0136
DEFINE strbyteptr =440700000001B#;                                       0137
DEFINE stringentity = entity=stringtype#;                                0138
DEFINE stringlength =22B8#;                                              0139
DEFINE stringmax = 2222B8#;                                              0140
DEFINE stringtype=4#;                                                    0141
DEFINE subop = 2#;                                                       0142
DEFINE suppressloc = rd.suploc#;                                         0143
DEFINE symbolicmode = 2#;                                                0144
DEFINE symbolproc = symtype.spaddr#;                                     0145
DEFINE symflg = db[18]#;                                                 0146
DEFINE textmode = 3#;                                                    0147
DEFINE typedefault=typemode=defaultmode#;                                0148
DEFINE typemode = fld.ftypmod#;                                          0149
DEFINE typenumeric = typemode=1#;                                        0150
DEFINE typeroutine=typemode#;                                            0151
DEFINE typespecial =typemode>1000#;                                      0152
DEFINE typesymbolic = typemode=2#;                                       0153
DEFINE typetext = typemode=3#;                                           0154
DEFINE undefaction = 0#;                                                 0155
DEFINE underflow=2#;                                                     0156
DEFINE underflowaction = rd.botact#;                                     0157
DEFINE wordbyteptr =44B8#;                                               0158
DEFINE wordentity = entity=wordtype#;                                    0159
DEFINE wordtype = 1#;                                                    0160
DEFINE xpointer = fieldptr.xptr=77B#;                                    0161
DEFINE xptraddr = fieldptr.xptadd/xptrbytesperword#;                     0162
DEFINE xptrbitpos =36-((fieldptr.xptadd MOD
xptrbytesperword)*fieldptr.xbytsz)#;                                     0163
```

```
   DEFINE xptrbytesperword=(36/fieldptr.xbytsz)#;                          0164
   DEFINE xptrbytesz = fieldptr.xbytsz#;                                   0165
   DEFINE xptrsize = fieldptr.xnbyte#;                                     0166
%                                                                          0167
%Defines%                                                                  0168
   %Miscellaneous%                                                         0169
      %Miscellaneous masks, I/O, values, instructions%                     0170
         DEFINE radix=8#;                                                  0171
         DEFINE global = 1#;                                               0172
         DEFINE gosig = 1#, continsig = 2#, quitsig = 3#;                  0173
         DEFINE noindexmask = 777760777777B#;                              0174
         DEFINE enterddt =!JSP 0,770002B#;                                 0175
         DEFINE nextchar = binjsys()#;                                     0176
         DEFINE backupchar = bkjfnjsys()#;                                 0177
         DEFINE addop = 1#, subop = 2#, mulop = 3#, divop = 4#;            0178
      %Hash table defines%                                                 0179
         DEFINE hashbase=19#, hashtblsz =300#;                             0180
         DEFINE hashtable=hsymtb#, hashstack=hstkhed#, hashptr
         =hsymptr#;                                                        0181
      %Byte pointers and fields%                                           0182
         DEFINE strbyteptr =440700000001B#;                                0183
         DEFINE recprecordaddr =22B8#, recprecsize =2222B8#;               0184
         DEFINE wordbyteptr =44B8#;                                        0185
         DEFINE stringmax = 2222B8#, stringlength =22B8#;                  0186
         DEFINE leftbyteptrrj = 4307B8#;                                   0187
      %Opcodes%                                                            0188
         DEFINE movei =201B#;                                              0189
         DEFINE jrst =254B9#;                                              0190
         DEFINE caml =311B9#, came =312B9#, camle=313B9#,
         camge=315B9#, camn =316B9#, camg =317B9#;                         0191
         DEFINE jsp = 265B9#;                                              0192
         DEFINE opcodmask =9M9#;                                           0193
         DEFINE opcall0=1#, opcall1=2#, opcallm=3#;                        0194
      %Sizes%                                                              0195
         DEFINE proctblsz = 10#, proctblentsz =2#;                         0196
         DEFINE maxwordsinstring = 402#;                                   0197
         DEFINE maxprms = 20#;                                             0198
         DEFINE bptblentsz = 4#, nbkpts=10#, bptblsz =40#;                 0199
         DEFINE ddtlimit = 500B#;                                          0200
         DEFINE optabl = 24B#;                                             0201
   %Record Information Block Defines (RIB)%                                0202
      %Recdesc defines%                                                    0203
         DEFINE recordstart = rd.raddr#;                                   0204
         DEFINE fieldnumber = rd.rindex#;                                  0205
         DEFINE numberfields = rd.nfield#;                                 0206
         DEFINE recordsize = rd.rsize#;                                    0207
         DEFINE mother = rd.momrec#;                                       0208
         DEFINE callmother=&calprc_mother; calprc#;                        0209
         DEFINE daughter = rd.kidrec#;                                     0210
         DEFINE recordname = rd.rdef#;                                     0211
         DEFINE replaceaction = rd.replact#;                               0212
         DEFINE defaultreplace = 1#;                                       0213
         DEFINE callreplaceproc = &calprc_rd.replact;calprc#;              0214
         DEFINE overflow=1#;                                               0215
         DEFINE undefaction = 0#, proceedaction = 1#;                      0216
         DEFINE overflowaction = rd.topact#;                               0217
```

```
        DEFINE fixoverflow =fixoutofbounds(&rd, overflowaction,
                                                                0218
        overflow)#;                                             0219
        DEFINE underflow=2#;                                    0220
        DEFINE underflowaction = rd.botact#;
        DEFINE fixunderflow =fixoutofbounds(&rd, underflowaction,
                                                                0221
        underflow)#;                                            0222
        DEFINE dynamicrecord = rd.rdproc#;                      0223
        DEFINE defineproc=rd.rdef#;                             0224
        DEFINE calldefineproc = &calprc_defineproc; calprc#;    0225
        DEFINE entity = rd.enttyp#;
        DEFINE wordtype = 1#, fieldtype=2#, recordtype=3#,
                                                                0226
        stringtype=4#;                                          0227
        DEFINE wordentity = entity=wordtype#;                   0228
        DEFINE fieldentity = entity=fieldtype#;                 0229
        DEFINE recordentity = entity=recordtype#;               0230
        DEFINE stringentity = entity=stringtype#;               0231
        DEFINE addressmode = rd.adrmod#;                        0232
        DEFINE macro = 0#, micro = 1#;                          0233
        DEFINE macromode = addressmode=macro#;                  0234
        DEFINE micromode =addressmode=micro#;                   0235
        DEFINE suppressloc = rd.suploc#;                        0236
        DEFINE initialised =rd.rdinit#;                         0237
        DEFINE rdsize = 6#; %Number of PDP10 words in record%   0238
    %Record Field Pointers%                                     0239
        DEFINE fieldstart = &rd+rdsize#;                        0240
        DEFINE fielddescsz=2#;
        DEFINE currentfld=&rd+rdsize+(IF fieldnumber >= numberfields
        THEN err($"Field Out of bounds") ELSE
        fieldnumber)*fielddescsz#;                              0241
        DEFINE setfieldptr =&fld _ currentfld#;                 0242
        DEFINE fieldptr=fld.fpointr#;                           0243
        DEFINE typemode = fld.ftypmod#;                         0244
        DEFINE defaultmode = 0#;                                0245
        DEFINE numericmode = 1#, symbolicmode = 2#, textmode = 3#;
                                                                0246
        DEFINE typedefault=typemode=defaultmode#;               0247
        DEFINE typenumeric = typemode=1#;                       0248
        DEFINE typesymbolic = typemode=2#;                      0249
        DEFINE typetext = typemode=3#;                          0250
        DEFINE typespecial =typemode>1000#;                     0251
        DEFINE typeroutine=typemode#;                           0252
        DEFINE calltyperoutine = &calprc_typeroutine; calprc#;  0253
    %X-pointers%                                                0254
        DEFINE xptrbytesperword=(36/fieldptr.xbytsz)#;          0255
        DEFINE xptraddr = fieldptr.xptadd/xptrbytesperword#;    0256
        DEFINE xptrbitpos =36-((fieldptr.xptadd MOD
        xptrbytesperword)*fieldptr.xbytsz)#;                    0257
        DEFINE xptrsize = fieldptr.xnbyte#;                     0258
        DEFINE xptrbytesz = fieldptr.xbytsz#;                   0259
        DEFINE xpointer = fieldptr.xptr=77B#;                   0260
    %Misz sizes, etc%                                           0261
        DEFINE maxnumfields = 50#;                              0262
        DEFINE ribsize = 106#; %rdsize + maxnumfields*fielddescsz%
                                                                0263
                                                                0264
%DDT Symbol value and type defines%                             0265
    DEFINE symbolproc = symtype.spaddr#;
```

```
      DEFINE stacksymbol = symtype.stksym#;                              0266
      DEFINE ddtsymbol = symtype.ddtsm#;                                 0267
      DEFINE sequencename = symtype.seqsym#;                             0268
      DEFINE fieldsymbol = symtype.fldsig#;                              0269
      DEFINE number = symtype.numsym#;                                   0270
      DEFINE programsymbol = symtype.progsym#;                           0271
      DEFINE specialsymbol = stacksymbol .V sequencename#;               0272
      DEFINE indirectsym = indddtsym#, indirectmode=1B6#;                0273
      DEFINE internalsym = 2B6#, internalfield = internalsym + 1B9#,
      internalstacksym = internalsym+1B6#;                               0274
      DEFINE seqsymbol = 4B6#, stacksymv =1B6#;                          0275
      DEFINE stackref=sequence#;                                         0276
   %Data Block Defines (DB)%                                             0277
      DEFINE reg1 = db#;                                                 0278
      DEFINE reg2 = db[1]#;                                              0279
      DEFINE reg3 = db[2]#;                                              0280
      DEFINE reg4 = db[3]#;                                              0281
      DEFINE reg5 = db[4]#;                                              0282
      DEFINE reg6 = db[5]#;                                              0283
      DEFINE reg7 = db[6]#;                                              0284
      DEFINE progp = db[7]#;                                             0285
      DEFINE progs = db[8]#;                                             0286
      DEFINE progm = db[9]#;                                             0287
      DEFINE progrp = db[10]#;                                           0288
      DEFINE rpreg = db[11]#;                                            0289
      DEFINE areg1 = db[12]#;                                            0290
      DEFINE areg2 = db[13]#;                                            0291
      DEFINE areg3 = db[14]#;                                            0292
      DEFINE areg4 = db[15]#;                                            0293
      DEFINE lv = db[16]#;                                               0294
      DEFINE escchar = db[17]#;                                          0295
      DEFINE symflg = db[18]#;                                           0296
      DEFINE framep = db[19]#;                                           0297
      DEFINE rfnmfg = db[20]#;                                           0298
      DEFINE currenttypemode = db[21]#;                                  0299
      DEFINE nparms = db[22]#;                                           0300
      DEFINE framesize = db[23]#;                                        0301
      DEFINE framebase = db[24]#;                                        0302
      DEFINE framebasetype = 2#;                                         0303
      DEFINE stacktop = db[25]#;                                         0304
      DEFINE lbptbl = db[26]#;                                           0305
      DEFINE lbptblsz = nbkpts#;                                         0306
      DEFINE lbptblentsz = 1#;                                           0307
      DEFINE lc = db[36]#;                                               0308
      DEFINE recp = db[37]#;                                             0309
      DEFINE ddttablesz = 38#;                                           0310
      DEFINE frametop =[stacktop].marklocfield#;                         0311
         DEFINE frametoptype =1#;                                        0312
      DEFINE returnloc =[framep].retlocfield#;                           0313
      DEFINE framemark =[framep].marklocfield#;                          0314
%Temporary stuff-- remove when integrating -- COMMENTED OUT==CHI          0315
   SET sin = 52B, bkjfn = 42B,atljb = 437B,jobtm = 424B, pbout = 74B,
   sout = 53B, pbin = 73B, kfork = 153B, nout = 224B, haltf = 170B,
   psout = 76B, gtjfn = 20B, openf = 21B, rljfn = 23B, closf = 22B,
   bin = 50B, bout = 51B, nin = 225B, rout = 55B, sizef = 36B, dtach
   = 115B, gjinf = 13B, stdir = 40B, login = 1, rpcap = 150B, epcap =
```

```
   151B, sir = 125B, ati = 137B, eir = 126B, aic = 131B, reset =
   147B, sfbsz = 46B, sevec = 204B;%                                    0316
REGISTER r0=0,r1 = 1, r2 = 2, r3 = 3, r4 = 4, r6 = 6, s = 11B, m =
12B, rp = 13B, a1 = 14B, a2 = 15B, a3 = 16B, a4 = 17B;                  0317
%Declarations%                                                          0318
   %Data Formats and meanings%                                          0319
      %Record Information Block (RIB)                                   0320
         There are two parts to a record information block:             0321
            (1) Record Descriptor                                       0322
               Identifies an entity generically                         0323
            Record Pointers...Point to th entity%                       0324
      (recdesc) RECORD %Record Descriptor...used for identifying  all
      entities used within DDT%                                         0325
         raddr[18] , %Address of start of data area%                    0326
         rindex[18], %Index into record (by field)%                     0327
         nfield[18], %Number of fields in record%                       0328
         rsize[18], %Record size in PDP10 Words%                        0329
         momrec[18], %Address of rd of mother record%                   0330
            %0 if none%                                                 0331
         kidrec[18], %address of inferior record%                       0332
            %0 if none%                                                 0333
         rdef[36], %Address of record definition%                       0334
         replact[18], %Action to take when replacing value%             0335
         topact[18], %Action for running off top of record%             0336
         botact[18], %Action for running off bottom (end)%              0337
         rdproc[1], %True if record defined by a procedure%             0338
         enttyp[6], %Type of entity at last display%                    0339
         adrmod[1], %Address Mode%                                      0340
         suploc[1], %Suress loc type flag%                              0341
         rdinit[1]; %True if record has been initialised%               0342
      (ribent)RECORD %Format of entry in RIB%                           0343
         fpointr[36], %byte or x-pointer to field%                      0344
         ttypmod[18]; %Type out mode indicator...%                      0345
      (xpnter)RECORD %A Special Format Byte Pnter used in RIB's%        0346
         xptadd[13], %Relative address in  bytes from start of
         record%                                                        0347
         xnbyte[11], %Numbr of bytes in field%                          0348
         xbytsz[6], %Byte Size%                                         0349
         xptr[6]; %Must be 77 for Byte Pointer to be xpointer%          0350
      %Symbol types, etc%                                               0351
         %Assumes symtype contains type as returned from getsym%        0352
         (symboltype)RECORD                                             0353
            spaddr[18], %Address of routine for parsing special
            symbols%                                                    0354
            stksym[1], %stack symbol%                                   0355
            ddtsm[1], %DDT Symbol%                                      0356
            seqsym[1], %Sequence name%                                  0357
            fldsig[1], %Field designator%                               0358
            numsym[1], %Number%                                         0359
            progsym[1]; %Program symbol name%                           0360
      %Getsym options%                                                  0361
         (symopt)RECORD                                                 0362
            ddtadr[1], %Return address of ddt symbol%                   0363
            escape[1]; %Escape flag%                                    0364
         DECLARE FIELD                                                  0365
            indddtsym=[0(rp), 1:18];                                   02766
```

```
   (bptblent)RECORD %Break Pont Table Entry%                          0366
      bpaddr[18], %Address of breakpoint%                             0367
      bpinst[36], %Instructio replaced by break%                      0368
      bptest[36], %Breakpoint test instruction                        0369
         =0: Unconditional break                                      0370
         1 = trace                                                    0371
         1-777777B =  test procedure address                          0372
         >777777B = Compare instruction                               0373
      %                                                               0374
      bpval[36]; %Value used with compare%                            0375
   DECLARE FIELD                                                      0383
      retlocfield= [0(rp), 18:0],                                    02762
      siglocfield= [0(rp), 18:18],                                    0384
      marklocfield= [-1(rp), 18:0];                                   0385
   % comment out -- move writeable data to DATA === CFD              0386
      DECLARE STRING ddtlit[150];                                     0387
      DECLARE bptbl[bptblsz]; %%nbkpts * bpentsz%%                    0388
      DECLARE EXTERNAL db;                                            0389
      DECLARE ddtrloc, ddtrp, ddtinst, ddtjunk, ddtrg1, ddtrg2,
      ddtrg3; %%Globals used by ddt during processing of
      breakpoints%%                                                   0390
      DECLARE hsymtb[hashtblsz], hstkhed[hashbase], hsymptr;          0391
      DECLARE levtbl, displayflag;                                    0392
      DECLARE proctbl[proctblsz];                                     0393
   === %                                                              0394
   DECLARE                                                           02761
      symloc= 116B,                                                  02763
      smanipumask= 77774B7,                                          02765
      smanipulator= 27044B7;                                         02764
   REF db;                                                           0396
%Control Procedures%                                                 0397
   (ddtcalls)PROC;                                                   0398
      %Various calling routines for ddt%                             0399
      %General register saving routine...saves 2-17B, assumes that 1(s)
      may be clobbered.  assumes r1 contains dest on call%           0400
         (ddtsvr1):                                                  0401
            !MOVEM 0, ddtrg3;                                        0402
            !MOVEI 0,16B(1); %Destination end%                       0403
            !HRLI 1,2;                                               0404
            !BLT 1,@0; %Do the move%                                 0405
            !JRST @ddtrg3; %RETURN%                                  0406
         (usvreg): %Save off the user registers, r1-r6 and a1-a4%    0407
            !MOVEM 0,ddtrg1;                                         0408
            !PUSH s,1;                                               0409
            !MOVEI 1,1(s);                                           0410
            !JSP ddtsvr1; %Registers on stack now%                   0411
            !POP s,0(s);                                             0412
            !HRRZ 1,db; %Destination%                                0413
            !MOVEI 0,5(1); %Dest%                                    0414
            !HRLI 1,1(s); %Source%                                   0415
            !BLT 1,@0; %r1 - r6%                                     0416
            areg1 _ [s+$a1];                                         0417
            areg2 _ [s+$a2];                                         0418
            areg3 _ [s+$a3];                                         0419
            areg4 _ [s+$a4];                                         0420
            !JRST @ddtrg1;                                           0421
```

```
      (urstrg):                                                          0422
         !MOVEM 0,ddtrg3;                                                0423
         r1.LH _ $areg1;                                                 0424
         r1.RH _ $a1;                                                    0425
         r0 _ $a4;                                                       0426
         !BLT 1,@0; %a1 - a4%                                            0427
         !MOVE 1,db;                                                     0428
         !HRLI 1,1(1);                                                   0429
         !HRRI 1,2;                                                      0430
         !MOVEI 0,6;                                                     0431
         !BLT 1,@0;                                                      0432
         !MOVE 1,@db;                                                    0433
         !JRST @ddtrg3;                                                  0434
      (calldt): %Garden variety call%                                    0435
         !SETZM 3(s); %Not called by breakpoint%                         0436
      (ddtcallcom):                                                      0437
         %Common call code...r0 contains return loc, 3(s) break parm%   0438
         !HRRZM 0,2(s); %Save off  eturn loc%                            0439
         !MOVEM 1,4(s); %Save off r1%                                    0440
         !MOVEI 1,5(s); %Destination for BLT%                            0441
         !JSP ddtsvr1; %Save 2-17%                                       0442
         !PUSH s,m;                                                      0443
         !PUSH s,1(s);                                                   0444
         m _ s;                                                          0445
         !PUSH s,1(s);                                                   0446
         GOTO runddt;                                                    0447
      (breakddt):                                                        0448
         %Breakpoiont call...top of stack contains break loc%            0449
         !POP s,2(s); %Move break loc to parm position%                  0450
         GOTO ddtcallcom;                                                0451
   END.                                                                  0452
(runddt)PROC(breakadr);                                                  0461
   LOCAL dblock[ddttablesz]; %Must be first thing on stack%               0462
   LOCAL xdb;                                                            0463
   LOCAL  c;                                                             0464
   LOCAL dspmode, dspsave;                                               0465
   LOCAL xnlmode, xbuffs, xbuffn;                                        0466
   LOCAL rd[ribsize];                                                    0467
   LOCAL i, tbuff[40];                                                   0468
   LOCAL xrawchr; % saved getchar dispatch %                             0469
   %Save off global values%                                              0470
      % save input buffer (buff) into temporary buffer (tbuff) %        0471
      FOR i _ 0 UP UNTIL > buffsz DO                                     0472
         tbuff[i] _ buff[i];                                             0473
      xbuffs _ buffs;                                                    0474
      xbuffn _ buffn;                                                    0475
      xdb _ &db;                                                         0476
   % save rawchar dispatch and supply normal one%                        0477
      %done so record/playback and auxinput stuff can use nddt%         0478
      xrawchr _ rawchr;  %save current away%                             0479
      rawchr _ $getchar; %and use system standard%                       0480
   %Initialise%                                                          0481
      % save away display area if in DNLS and no tty simulation area
      has been allocated %                                               0482
      %Save away nlmode; set it to typewriter for proper input%         0483
```

```
      IF (xnlmode _ nlmode) = fulldisplay THEN                          0484
         BEGIN                                                          0485
         IF (tenex > 13300) AND (nldevice = devlproc) THEN              02746
            BEGIN                                                       02747
            !bout( dspjfn, lpesc );                                     02748
            !bout( dspjfn, lpnocoor);                                   02749
            END;                                                        02750
         IF defttysim THEN                                              0486
            BEGIN % dnls and no tty simulation area %                   0487
            IF tenex < 13200 THEN !JSYS 451B; % tsnda %                 0488
            displayflag _ FALSE;                                        0489
            dspsave _ 2;                                                0490
            END                                                         0491
         ELSE                                                           0492
            BEGIN % dnls and tty simulation area %                      0493
            displayflag _ TRUE;                                         0494
            dspsave _ 1;                                                0495
            END;                                                        0496
         % set control character output codes %                         0497
            r1 _ 18M;                                                   0498
            r2 _ ttycoc;                                                0499
            r3 _ 1B3;                                                   0500
            !JSYS sfcoc;                                                0501
         END                                                            0502
      ELSE                                                              0503
         BEGIN % tty device %                                           0504
         displayflag _ FALSE;                                           0505
         dspsave _ FALSE;                                               0506
         END;                                                           0507
      nlmode _ typewriter;                                              0508
   &db _ $dblock;                                                       0509
   %When called, dblock [0-14] contains registers 1-17%                 0510
   BUMP rubabt; %For stopping loops  n debugging mode%                  0511
   %Set up frame pointer%                                               0512
      stacktop _ m.RH;                                                  0513
      framebase _ $gstack+2;                                            0514
      framep _ frametop;                                                0515
      initframe();                                                      0516
   %Set up other stuff%                                                 0517
      recp _ rpreg;                                                     0518
      escchar _ '; ;                                                    0519
      symflg _ TRUE;                                                    0520
      rfnmfg _ TRUE; % turn on printing of names of records %           0521
      lv _ 0;                                                           0522
      lc _ 0;                                                           0523
      c _ -1;                                                           0524
      WHILE (c_c+1) <= lbptblsz DO [$lbptbl+c] _ 0;                     0525
   %Restore all breakpoints, and type message if called from
   breakpoint.  Notice that in order to use breakpoints in nddt,
   you must call nddt from nddt (^H), set the breakpoint, and then
   return with a continue%                                              0526
      restbp();                                                         0527
      IF breakadr THEN                                                  0528
         BEGIN                                                          0529
         typbrk(breakadr-1);                                            0530
         END;                                                           0531
```

```
                                                                    0532
    %Initialise Symbol Table%                                       0533
       initht();                                                    0534
 %Now start parsing%                                                0535
    clrbuf(0);                                                      0536
    typeas($"NLS DDT");                                             0537
    LOOP                                                            0538
       BEGIN                                                        0539
       ON SIGNAL                                                    0540
          =quitsig:                                                 0541
             IF breakadr THEN callsig(continsig)                    0542
             ELSE                                                   0543
                BEGIN
                % set sysmsg to 0 so that it won't fake out the
                                                                    0544
                parser %                                            0545
                   sysmsg _ 0;                                      0546
                %Restore globals%                                   0547
                rawchr _ xrawchr;                                   0548
                &db _ xdb;
                FOR i _ 0 UP UNTIL > buffsz DO buff[i] _ tbuff[i];
                                                                    0549
                                                                    0550
                buffn _ xbuffn;                                     0551
                buffs _ xbuffs;                                     0552
                nlmode _ xnlmode;                                   0553
                IF dspsave THEN                                     0554
                   BEGIN                                            0555
                   % set control character output codes %           0556
                      r1 _ 18M;                                     0557
                      r2 _ dpycoc;                                  0558
                      r3 _ 1B3;                                     0559
                      !JSYS sfcoc;
                   %  restore DNLS display area if it was saved %
                                                                    0560
                      IF (tenex < 13200) THEN                       0561
                                                                   02752
                         BEGIN
                         IF dspsave = 2 THEN !JSYS 452B; %tsfda%
                                                                   02751
                                                                   02753
                         END
                      ELSE IF nldevice = devlproc THEN lpcmode();
                                                                   02754
                                                                    0563
                   END;                                             0564
                RETURN;                                             0565
                END;                                                0566
          =continsig:                                               0567
             BEGIN                                                  0568
             %Restore globals%
             % reset sysmsg so system won't think it has to put out
                                                                    0569
             an error message %                                     0570
                sysmsg _ 0;                                         0571
             rawchr _ xrawchr;                                      0572
             &db _ xdb;
             FOR i _ 0 UP UNTIL > buffsz DO buff[i] _ tbuff[i];
                                                                    0573
                                                                    0574
             buffn _ xbuffn;                                        0575
             buffs _ xbuffs;                                        0576
             nlmode _ xnlmode;                                      0577
             IF dspsave THEN
```

```
            BEGIN                                                   0578
            % set control character output codes %                  0579
               r1 _ 18M;                                            0580
               r2 _ dpycoc;                                         0581
               r3 _ 1B3;                                            0582
               !JSYS sfcoc;                                         0583
            %  restore DNLS display area if it was saved %
                                                                    02755
               IF (tenex < 13200) THEN                              02756
                  BEGIN                                             02757
                  IF dspsave = 2 THEN !JSYS 452B; %tsfda%           02758
                  END                                               02759
               ELSE IF nldevice = devlproc THEN lpcmode();
                                                                    02760
            END;                                                    0587
         !MOVEI 17B; %Last word to move into%                       0588
         !HRLZI 1,3(m); %From%                                      0589
         !HRRI 1,2; %to%                                            0590
         !BLT 1,@0; %xfer regs 2-17%                                0591
         %Note that s now points to the proper m-2 for the
         frame we are actually in, and m has been restored to
         previous frame%                                            0592
         !MOVE 1,4(s); %Restore register 1%                         0593
         !HRRZ 0,2(s); %Return loc%                                 0594
         !JRST @0; %RETURN%                                         0595
         END;                                                       0596
      =gosig:                                                       0597
         BEGIN                                                      0598
         %Sysmsg contains stat location%                            0599
         %fix up breakad and fake a return from a breakpoint%
                                                                    0600
         breakadr _ sysmsg+1; %One past first instruction is
         proper return loc%                                         0601
         [m].retlocfield _ $breakret;                               0602
         callsig(contsinsig);                                       0603
         END;                                                       0604
      =statesig: REPEAT LOOP;                                       0605
      ELSE                                                          0606
         BEGIN                                                      0607
         IF sysmsg > 1000 THEN                                      0608
            BEGIN                                                   0609
            crlf();                                                 0610
            typeas(sysmsg);                                         0611
            END;                                                    0612
         REPEAT LOOP;                                               0613
         END;                                                       0614
   crlf();                                                          0615
   typech(">);                                                      0616
   echoff();                                                        0617
   CASE nextchar OF                                                 0618
      ="B:  %Breakpoint set, clear, print%                          0619
         xbrkpt();                                                  0620
      ="C:  %Continue program -- return from ^H%                    0621
         xcontin();                                                 0622
      ="D: % Define symbol %                                        0623
         xdddef();                                                  0624
```

```
            ="F:  % find content %                                          0625
               xddfind();                                                   0626
            ="G: %go to location%                                          0627
               xgo();                                                       0628
            ="M:  % mark symbol table %                                    0629
               xnmark();                                                    0630
            ="P:  %Procedure call, replace, backup%                        0631
               xprocc();                                                    0632
            ="R:  %Record Pointer Set%                                     0633
               xrpset();                                                    0634
            ="S:  %Show Record, String, Location, Preceeding, Next%
                                                                           0635
               xnshow($rd, 1);                                              0636
            ="T:   %Trace%                                                 0637
               xtrace();                                                    0638
            ="V:  %Value of symbol%                                        0639
               xtval();                                                     0640
            ="=: %type numeric value of symbol%                            0641
               BEGIN                                                        0642
               typech("=);                                                  0643
               typval(lv, numericmode);                                     0644
               END;                                                         0645
            =LF,  %show next%                                              0646
            ="_,   % assign a value %                                      0647
            ="^,  %show previous%                                          0648
            =TAB:  %show value of current symbol%                          0649
               BEGIN                                                        0650
               backupchar;                                                  0651
               xnshow($rd, 0);                                              0652
               END;                                                         0653
            ENDCASE typeas($" ??");                                        0654
         END;                                                              0655
      RETURN;                                                              0656
      END.                                                                 0657
%* Low Level syntactic recognizers %                                       0658
   (chkcacr)PROC;                                                          0659
      LOCAL char;                                                          0660
      char _ nextchar;                                                     0661
      IF char # CA AND char # EOL THEN err($"?? CA Expected");              0662
      RETURN;                                                              0663
      END.                                                                 0664
   (deblnk)PROC;                                                           0665
      WHILE nextchar = SP DO NULL;                                         0666
      backupchar;                                                          0667
      RETURN;                                                              0668
      END.                                                                 0669
   (rdnum)PROC( base);                                                     0680
      LOCAL char, bp, v;                                                   0681
      LOCAL STRING tempsr[20];                                             0682
      %Read a number, and return value plus number if ok%                  0683
      IF (char _ nextchar) IN ["0, "9] OR char = "-   THEN                 0684
         BEGIN                                                             0685
         *tempsr* _ char;                                                  0686
            LOOP                                                           0687
               BEGIN                                                       0688
               CASE char _ nextchar OF                                     0689
```

```
                =BC:                                                    0690
                   IF tempsr.L > 0 THEN                                 0691
                      BEGIN                                             0692
                      typech( '\ );                                     0693
                      typech( *tempsr*[tempsr.L] ); % echo last
                      character %                                       0694
                      tempsr.L _ tempsr.L-1;                            0695
                      END;                                              0696
                =8W, =21B: RETURN( rdnum(base) );                       0697
                IN ['0, '9]: *tempsr* _ *tempsr*, char;                 0698
                ENDCASE                                                 0699
                   BEGIN                                                0700
                   backupchar;                                          0701
                   bp _ strbyteptr + $tempsr;                           0702
                   *tempsr* _ *tempsr*, 0;                              0703
                   IF NOT ninjsys($bp, base: v) THEN err($"Illegal
                   Number");                                            0704
                   RETURN(TRUE, v);                                     0705
                   END;                                                 0706
             END;                                                       0707
          END;                                                          0708
       backupchar;                                                      0709
       RETURN(FALSE);                                                   0710
       END.                                                             0711
    (rdsym)PROC(astr);                                                  0712
       LOCAL char;                                                      0713
       REF astr;                                                        0714
       %Read a symbol into astr, returning false if none%               0715
       *astr* _ NULL;                                                   0716
       IF (char _ nextchar) IN ['A, 'Z] THEN                            0717
          LOOP                                                          0718
             BEGIN                                                      0719
             CASE char OF                                               0720
                =BC:                                                    0721
                   IF astr.L > 0 THEN                                   0722
                      BEGIN                                             0723
                      typech( '\ );                                     0724
                      typech( *astr*[astr.L] ); % echo last character %
                                                                        0725
                      astr.L _ astr.L-1;                                0726
                      END;                                              0727
                =8W, =21B: astr.L _ 0;                                  0728
                IN ['A, 'Z], IN ['0, '9]: *astr* _ *astr*, char;        0729
                ENDCASE                                                 0730
                   BEGIN                                                0731
                   backupchar;                                          0732
                   RETURN(astr.L);                                      0733
                   END;                                                 0734
             char _ nextchar;                                           0735
             END;                                                       0736
       backupchar;                                                      0737
       RETURN(FALSE);                                                   0738
       END.                                                             0739
    %* Typeout routines %                                              0740
       (trfname)PROC(bp, rd);                                           0741
          %Types out a field of a record with the appropriate name%     0742
```

```
   LOCAL STRING tempsr[40];                                              0743
   REF rd;                                                               0744
   ddgtsymbol($tempsr, recordname.RH + fieldnumber);                     0745
   crlf();                                                               0746
   typeas($tempsr);                                                      0747
   IF currenttypemode = numericmode THEN typech("[)                      0748
   ELSE IF currenttypemode = textmode THEN typech("")                    0749
   ELSE typech("/);                                                      0750
   typval(.bp, currenttypemode);                                         0751
   RETURN;                                                               0752
   END.                                                                  0753
(typentity)PROC(rd, curmode);                                            0754
   %Types out the entity described by rd in appropriate manner%          0755
   REF rd;                                                               0756
   ic _ recordstart;                                                     0757
   IF micromode THEN typfield(&rd, curmode)                              0758
   ELSE                                                                  0759
      IF numberfields = 0 THEN                                           0760
         BEGIN                                                           0761
         crlf();                                                         0762
         typeas($"Record Empty");                                        0763
         END                                                             0764
      ELSE                                                               0765
         BEGIN                                                           0766
         fieldnumber _ 0;                                                0767
         WHILE fieldnumber < numberfields DO                             0768
            BEGIN                                                        0769
            typfield(&rd, curmode);                                      0770
            fieldnumber _ fieldnumber + 1                                0771
            END;                                                         0772
         END;                                                            0773
   RETURN;                                                               0774
   END.                                                                  0775
(typfield)PROC(rd, curmode);                                             0776
   %rd = address of Record Descriptor, curmode = default mode%           0777
   LOCAL index, fld, mode, t, bp, calprc, count;                         0778
   REF rd, fld, calprc;                                                  0779
   setfieldptr;                                                          0780
   IF typespecial THEN                                                   0781
      BEGIN                                                              0782
      calltyperoutine((fieldptr + recordstart) .A noindexmask, &rd);
                                                                         0783
      END                                                                0784
   ELSE                                                                  0785
      BEGIN %Type otherwise%                                             0786
      mode _ IF typedefault THEN curmode ELSE typemode;                  0787
      IF xpointer THEN                                                   0788
         BEGIN                                                           0789
         bp _ 0;                                                         0790
         count _ xptrsize;                                               0791
         bp.bpadr _ recordstart + xptraddr;                              0792
         bp.bpbitpos _ xptrbitpos;                                       0793
         bp.bpsize _ xptrbytesz;                                         0794
         WHILE (count _ count - 1) > -1 DO                               0795
            typval(^bp, mode);                                           0796
         END                                                             0797
```

```
      ELSE                                                              0798
         BEGIN %An ordinary poiter%                                     0799
         bp _ (fieldptr + recordstart) .A noindexmask;                  0800
         typval(.bp, mode);                                             0801
         END;                                                           0802
      END;                                                              0803
   RETURN;                                                              0804
   END.                                                                 0805
(typloc)PROC(rd);                                                       0806
   LOCAL fld, mode;                                                     0807
   LOCAL STRING tempsr[75];                                             0808
   REF rd, fld;                                                         0809
   IF suppressloc THEN RETURN;                                          0810
   setfieldptr;                                                         0811
   mode _ IF typedefault THEN currenttypemode ELSE typemode;            0812
   IF mode = numericmode THEN *tempsr* _ *[octnum(recordstart)]*        0813
   ELSE ddgtsymbol($tempsr, recordstart);                               0814
   typeas($tempsr);                                                     0815
   IF NOT typespecial THEN                                              0816
      BEGIN                                                             0817
      IF mode =  numericmode THEN typech('[)                            0818
      ELSE IF mode = textmode THEN typech('")                           0819
      ELSE typech('/);                                                  0820
      END;                                                              0821
   RETURN;                                                              0822
   END.                                                                 0823
%* Character I/O Routines %                                            0845
(binjsys)PROC;                                                          0846
   LOCAL char;                                                          0847
   RETURN(CASE (char _ inpcuc()) OF                                     0848
      =CD: callsig(statesig);                                           0849
      ENDCASE char);                                                    0850
   END.                                                                 0851
(bkjfnjsys)PROC;                                                        0852
   IF (buffs _ buffs - 1) < 0 THEN buffs _ buffsz;                      0853
   RETURN (buff[ buffs ]);                                              0854
   END.                                                                 0855
(ninjsys)PROC(inpbp, base);                                             0856
   LOCAL rflag, char, v;                                                0857
   REF inpbp;                                                           0858
   rflag _ 0;                                                           0859
   r1 _ inpbp;                                                          0860
   r3 _ base;                                                           0861
   IF NOT SKIP !JSYS nin THEN rflag _ TRUE;                             0862
   inpbp _ r1;                                                          0863
   RETURN(NOT rflag, r2);                                               0864
   END.                                                                 0865
%* Parsing Routines - in Alphabetical order by command name %          0866
   %* Addresses and values%                                            0867
   (chngadr)PROC(rd,value);                                             0868
      REF rd;                                                           0869
      IF NOT initialised THEN err($"Relative addressing attempted on
      unitialised entity");                                             0870
      IF value = 0 THEN RETURN;                                         0871
      IF macromode THEN                                                 0872
         BEGIN                                                          0873
```

```
      WHILE value DO                                                    0874
         IF value < 0 THEN                                              0875
            BEGIN                                                       0876
            fixunderflow;                                               0877
            value _ value + 1;                                          0878
            END                                                         0879
         ELSE                                                           0880
            BEGIN                                                       0881
            fixoverflow;                                                0882
            value _ value - 1;                                          0883
            END;                                                        0884
      addressmode _ macro;                                              0885
      END                                                               0886
   ELSE                                                                 0887
      BEGIN                                                             0888
      IF (fieldnumber _ fieldnumber + value) NOT IN [0,
      numberfields] THEN                                                0889
         BEGIN                                                          0890
         IF fieldnumber < 0 THEN fixunderflow                           0891
         ELSE fixoverflow;                                              0892
         addressmode _ micro;                                           0893
         END;                                                           0894
      END;                                                              0895
   RETURN;                                                              0896
   END.                                                                 0897
(fielddesig)PROC(rd);                                                   0898
   LOCAL value, fld, symtype, symval;                                   0899
   REF rd, fld;                                                         0900
   IF NOT wordentity AND NOT fieldentity THEN RETURN; %Strings AND
   records may not be followed by fields%                               0901
   deblnk();                                                            0902
   IF nextchar # '. THEN                                                0903
      BEGIN                                                             0904
      backupchar;                                                       0905
      RETURN;                                                           0906
      END;                                                              0907
   deblnk();                                                            0908
   IF NOT getsym( :symtype, symval)                                     0909
   AND NOT programsymbol                                                0910
   AND NOT fieldsymbol THEN                                             0911
      err($"Illegal Symbol in Address");                                0912
   IF NOT ckbptr(value _ [symval]) THEN err($"Illegal Field
   Designator");                                                        0913
   setfieldptr;                                                         0914
   IF wordentity THEN fieldptr.bpadr _ value.bpadr                      0915
   ELSE                                                                 0916
      IF value.bpadr # 0 THEN err($"Illogical Concatenation of
      fields")                                                          0917
      ELSE value.bpadr _ fieldptr.bpadr;                                0918
   IF fieldptr.bpsize <= value.bpbitpos                                 0919
   OR (value.bpbitpos _ value.bpbitpos + fieldptr.bpbitpos) > 35
                                                                        0920
   OR (value.bpsize _ MIN(fieldptr.bpsize, value.bpsize,
   (fieldptr.bpbitpos _ fieldptr.bpsize) - value.bpbitpos)) <= 0
   THEN err($"Illogical concatenation of fields");                      0921
   value.bpindx _ 0;                                                    0922
```

```
    entity _ fieldtype;                                                 0923
    fieldptr _ value;                                                   0924
    fielddesig(&rd);                                                    0925
    RETURN;                                                             0926
    END.                                                                0927
(fixoutofbounds)PROC(rd, action, direction);                            0928
    LOCAL calprc;                                                       0929
    REF rd, calprc;                                                     0930
    IF action = undefaction THEN err($"Address Out Of Bounds");
                                                                        0931
    IF action = proceedaction THEN                                      0932
        recordstart _ recordstart + (IF direction = overflow THEN
        recordsize ELSE -recordsize)                                    0933
    ELSE                                                                0934
        BEGIN                                                           0935
        &calprc _ action;                                               0936
        calprc(&rd);                                                    0937
        END;                                                            0938
    initialised _ FALSE;                                                0939
    initrd(&rd);                                                        0940
    RETURN;                                                             0941
    END.                                                                0942
(getop)PROC;                                                            0943
    RETURN( CASE nextchar OF                                            0944
        =SP, ="+: addop;                                                0945
        ="-: subop;                                                     0946
        ="*: mulop;                                                     0947
        ="/: divop;                                                     0948
        ENDCASE IF backupchar THEN 0 ELSE 0);                           0949
    END.                                                                0950
(getsym)PROC;                                                           0951
    LOCAL option, symtype, symval, char, v1, v2, modesave, bp;          0952
    LOCAL STRING tempsr[30];                                            0953
    %Check for escape, etc%                                             0954
        deblnk();                                                       0955
        option _ 0;                                                     0956
        CASE char _ nextchar OF                                         0957
            = "=:                                                       0958
                BEGIN                                                   0959
                option.ddtadr _ 1; %DDT symbol return rela address not
                contents%                                               0960
                REPEAT CASE;                                            0961
                END;                                                    0962
            = escchar:                                                  0963
                BEGIN                                                   0964
                option.escape _ 1; %escape character%                   0965
                REPEAT CASE;                                            0966
                END;                                                    0967
            ENDCASE;                                                    0968
    symtype _ symval _ 0;                                               0969
    backupchar;                                                         0970
    IF (char = CA) AND displayflag THEN % got a bug %                   0971
        BEGIN                                                           0972
        *tempsr* _ NULL;                                                0973
        modesave _ nlmode; % fake out display mode %                    0974
        nlmode _ 1; % full display %                                    0975
```

```
        LOOP                                                              0976
           BEGIN                                                          0977
           INPUT NAME tempsr;                                             0978
           CASE char _ nextchar OF                                        0979
              =BC:                                                        0980
                 BEGIN                                                    0981
                 bmoff(); dn($"  "); % reset stuff %                      0982
                 REPEAT LOOP;                                             0983
                 END;                                                     0984
           ENDCASE                                                        0985
              BEGIN                                                       0986
              backupchar;                                                 0987
              EXIT LOOP;                                                  0988
              END;                                                        0989
           END;                                                           0990
        bmoff(); dn($"  "); % reset stuff %                              0991
        nlmode _ modesave;                                                0992
        echon();                                                          0993
        echo( $tempsr ); % echo it %                                      0994
        IF char # CA THEN typech(char);                                   0995
        IF tempsr.L > 0 THEN                                              0996
           IF *tempsr*[1] IN ['0, '9] THEN                                0997
              BEGIN % got a number%                                       0998
              bp _ strbyteptr + $tempsr;                                  0999
              *tempsr* _ *tempsr*, 0;                                    01000
              IF NOT ninjsys($bp, radix: symval) THEN err($"Illegal
              Number");                                                  01001
              GOTO gotnum;                                               01002
              END                                                        01003
           ELSE GOTO gotsym;                                             01004
        END;                                                              01005
     IF rdnum( radix: symval) THEN                                        01006
        BEGIN %Number%                                                    01007
        (gotnum):                                                         01008
        number _ TRUE;                                                    01009
        IF nextchar = ', THEN %Possible Half word Format%                 01010
           IF nextchar = ', THEN                                          01011
              BEGIN                                                       01012
              symval.LH _ symval;                                         01013
              IF NOT rdnum( radix: v1) THEN err($"Illegal Number");
                                                                         01014
              symval.RH _ v1;                                            01015
              END                                                        01016
           ELSE BEGIN                                                    01017
           backupchar; % back over non-comma %                           01018
           backupchar; % back over comma %                               01019
           END                                                           01020
        ELSE backupchar; % back over non-comma %                          01021
        END                                                               01022
     ELSE                                                                 01023
        BEGIN                                                             01024
        IF NOT rdsym($tempsr) THEN RETURN(FALSE);                         01025
        (gotsym):                                                         01026
        IF symflg .X option.escape # 0 THEN %Check as a DDT symbol%
                                                                         01027
           IF ddtsym($tempsr: symtype, v2) THEN                          01028
```

```
            BEGIN                                                   01029
            IF ddtsymbol THEN %Simple ddt internal symbol%          01030
               BEGIN                                                01031
               v2 _ [v2]+$reg1; %Actual address for this version%
                                                                    01032
               symval _ IF v2.indirectsym THEN                      01033
                  IF option.ddtadr THEN v2.RH                       01034
                  ELSE [v2.RH]                                      01035
               ELSE v2.RH;                                          01036
               END                                                  01037
            ELSE symval _ v2;                                       01038
            GOTO comsym;                                            01039
            END;                                                    01040
         IF option.ddtadr THEN err($"Not DDT Symbol after '="");    01041
         %Now do a ddt lookup%                                      01042
            IF NOT ddtlookup($tempsr, option.escape: symval) THEN
                                                                    01043
               BEGIN                                                01044
               *ddtlit* _ "No Such Symbol ", *tempsr*;              01045
               err($ddtlit);                                        01046
               END;                                                 01047
            programsymbol _ TRUE;                                   01048
         END;                                                       01049
      %Common symbol lookup stuff%                                  01050
         (comsym):                                                  01051
      RETURN(TRUE, symtype, symval);                                01052
      END.                                                          01053
   (oper)PROC(value, value1, op);                                   01054
      value _ CASE op OF                                            01055
         =addop: value+value1;                                      01056
         =subop: value-value1;                                      01057
         =mulop: value*value1;                                      01058
         =divop: value/value1;                                      01059
         ENDCASE err($"Illegal Operator");                          01060
      RETURN(value);                                                01061
      RETURN;                                                       01062
      END.                                                          01063
   (parsenter) PROCEDURE;                                           01064
      % this routine parses the current input stream looking for a
      value being typed in -                                        01065
      It returns: FALSE if no value is typed in                     01066
                  TRUE and the value if given %                     01067
      LOCAL value, field, i, char;                                  01068
      LOCAL STRING tempstr[10];                                     01069
      % look at the next source char but do not read it yet %       01070
      char _ nextchar;                                              01071
         backupchar;                                                01072
      % process according to what was typed in %                    01073
      CASE char OF                                                  01074
         =CA, =C., =CR:  % nothing given %                          01075
            RETURN (FALSE);                                         01076
         IN ['A, 'Z]:  % start of a symbol %                        01077
            BEGIN                                                   01078
            rdsym( $tempstr ); % gather symbol %                    01079
            % find out if it is a PDP10 instruction mnemonic %     01080
            IF NOT( value_ insym($tempstr)) THEN                    01081
```

```
            % not mnemonic -- return symbol to input buffer %
                                                                    01082
            FOR i _ 1 UP UNTIL > tempstr.L DO backupchar            01083
                                                                    01084
         ELSE LOOP                                                  01085
            BEGIN  % parse the PDP10 instruction %
            % get next field - perform range check to trap out
            values for the symbolic register names %                01086
            IF (field _ parsevalue(0,0,0)) IN [$reg1, $areg4] THEN
                                                                    01087
            field _ field - $reg1 + 1;                              01088
            CASE char _ nextchar OF                                 01089
               =CA, =C., =CR:  % all done %                         01090
                  BEGIN                                             01091
                  value.addr10 _ field;                             01092
                  backupchar;                                       01093
                  RETURN (TRUE, value);                             01094
                  END;                                              01095
               ='(:  % an accumulator field was given %             01096
                  value.accum10 _ field;                            01097
               ='@:  % indirect address %                           01098
                  value.indir10 _ TRUE;
               ='(:  % index coming up -- last field must be addr
                                                                    01099
               %                                                    01100
                  value.addr10 _ field;                             01101
               ='):  % got index %                                  01102
                  BEGIN                                             01103
                  value.index10 _ field;                            01104
                  field _ value.addr10;                             01105
                  REPEAT CASE;                                      01106
                  END;                                              01107
            ENDCASE err($"Illegal Instruction Format");             01108
            END; % of parse loop %                                  01109
         END; % of symbol case %                                    01110
      ='": %start of a string %                                     01111
         BEGIN                                                      01112
         *ddtlit* _ NULL; % use ddtlit to collect string %          01113
         nextchar; % read over the initial " %                      01114
         LOOP                                                       01115
            CASE char _ nextchar OF                                 01116
               =BC: ddtlit.L _ MAX(ddtlit.L-1,0);                   01117
               ='": EXIT LOOP; % read terminating " %               01118
            ENDCASE *ddtlit* _ *ddtlit*, char;                      01119
         RETURN (TRUE, $ddtlit);                                    01120
         END;                                                       01121
ENDCASE;
IF (value _ parsevalue(0,0,0)) IN [$reg1, $areg4] THEN value _
                                                                    01122
value - $reg1 + 1;                                                  01123
RETURN (TRUE, value);                                               01124
END.                                                                01125
(insym) PROCEDURE( symbol );
   % this routine identifies the given symbol as a PDP10
   instruction and returns the skeleton for the instruction if
                                                                    01126
   found, FALSE otherwise %                                         01127
   LOCAL sixbit, i, value;                                          01128
   REF symbol;                                                      01129
   % range check on length of symbol %
   IF NOT (symbol.L IN [1, 6]) THEN RETURN (FALSE);                 01130
```

```
        sixbit _ 0;                                                    01131
        value _ 0;                                                     01132
        % convert symbol to sixbit form - left adjusted %              01133
        FOR i _ 1 UP UNTIL > 6 DO                                      01134
           sixbit _ shl(sixbit,6) +                                    01135
              (IF i <= symbol.L THEN*symbol*[i] - 32 ELSE 0);          01136
        % try to match sixbit value with optab entries %              01137
        FOR i _ 1 UP UNTIL >= 700B DO % range of opcodes %             01138
           IF optab[i] = sixbit THEN                                   01139
              BEGIN                                                    01140
              value.opcod10 _ i;                                       01141
              RETURN (value);                                          01142
              END;                                                     01143
        % try aux table for long opcodes %                            01144
        FOR i _ 0 UP UNTIL >= optab1 DO                                01145
           IF optab3[i] = sixbit THEN                                  01146
              BEGIN                                                    01147
              value.longopcode _ optab2[i];                            01148
              RETURN (value);                                          01149
              END;                                                     01150
        % no can find-- give up %                                      01151
        RETURN (FALSE);                                                01152
        END.                                                           01153
    (parsevalue)PROC(rd, symtype, symval);                             01154
       LOCAL op, value, defvalue;                                      01155
       REF rd;                                                         01156
       IF symtype = 0 THEN %Get the first symbou%                      01157
          IF defvalue _ NOT getsym(: symtype, symval) THEN             01158
             BEGIN                                                     01159
             value _ IF &rd THEN recordstart ELSE 0;                   01160
             GOTO parsewrapup;                                         01161
             END;                                                      01162
       IF specialsymbol THEN err($"Illegal Address");                  01163
       value _ symval;                                                 01164
       LOOP                                                            01165
          BEGIN                                                        01166
          IF (op _ getop()) = 0 THEN EXIT LOOP;                        01167
          IF (NOT getsym( :symtype, symval)) OR specialsymbol THEN
          err($"Illegal Address");                                     01168
          value _ oper(value, symval, op);                            01169
          END;                                                         01170
       (parsewrapup):                                                  01171
       IF &rd THEN                                                     01172
          BEGIN                                                        01173
          recordstart _ value;                                         01174
          initrd(&rd); %Iniialise record%                              01175
          fielddesig(&rd);                                             01176
          END;                                                         01177
       RETURN(value, defvalue);                                        01178
       END.                                                            01179
    %Replace%                                                          01180
       (replfield)PROC(rd);                                            01181
          %This procedure accepts an expression, and replaces the
          contents of the field indicated by rd with its value.  It
          expects the expression to be terminated with a CA.%          01182
          LOCAL value, char;                                           01183
```

```
        REF rd;                                                          01184
        IF macromode THEN                                                01185
           IF numberfields = 1 THEN fieldnumber _ 0                      01186
           ELSE err($"Illegal Replace")                                  01187
        ELSE                                                             01188
           IF fieldnumber >= numberfields THEN err($"Illegal
           Replace");                                                    01189
        IF (char _ nextchar) = CA OR char = EOL THEN RETURN; %Empty%     01190
                                                                         01191
        backupchar;                                                      01192
        IF parsenter( :value) THEN setfield(&rd, value);                 01193
        chkcacr();                                                       01194
        RETURN;                                                          01195
        END.                                                             01196
     (replrecord)PROC(rd);
        %REplace the record indicated by rd with a series of
        values::                                                         01197
           EMPTY CA means don't change it                                01198
           Value CA means replace it with value                          01199
           %                                                             01200
        %For each field, issue CRLF and type name of field%              01201
        LOCAL value, fld;                                                01202
        LOCAL STRING tempsr[40];                                         01203
        REF rd, fld;                                                     01204
        IF NOT initialised THEN err($"Illegal Replace");                 01205
        fieldnumber _ 0;                                                 01206
        crlf();                                                          01207
        LOOP                                                             01208
           BEGIN                                                         01209
           setfieldptr;                                                  01210
           %First get name of field, and type it%                        01211
              IF NOT dynamicrecord THEN                                  01212
                 BEGIN                                                   01213
                 ddgtsymbol($tempsr, (recordname + fieldnumber) .A
                 18M);                                                   01214
                 typeas($tempsr);                                        01215
                 typeas($" _ ");                                         01216
                 END;                                                    01217
           %Now get value if thre is one%                                01218
              CASE nextchar OF                                           01219
                 =CA, =CR: NULL; %Ignore this field%                     01220
                 ENDCASE                                                 01221
                    BEGIN                                                01222
                    backupchar;                                          01223
                    IF parsenter( :value) THEN setfield(&rd, value);
                                                                         01224
                    chkcacr(); % read over CA or CR %                    01225
                    END;                                                 01226
           IF (fieldnumber _ fieldnumber + 1) = numberfields THEN
                                                                         01227
           EXIT;                                                         01228
           END;                                                          02769
        crlf();                                                          01229
        typeas($"End of Record");                                        01230
        RETURN;                                                          01231
        END.                                                             01232
     (repval)PROC(rd);
```

```
        LOCAL calprc, char;                                          01233
        REF rd, calprc;                                              01234
        %Parse last part of show command, and replace value as
        required%                                                    01235
        echoff();                                                    01236
        IF (char _ nextchar) # "_ THEN                               01237
           BEGIN                                                     01238
           IF char = "[ OR char = "/ THEN RETURN ;                   01239
           backupchar;                                               01240
           chkcacr();                                                01241
           RETURN;                                                   01242
           END;                                                      01243
        typeas($" _ ");                                              01244
        echon();                                                     01245
        IF replaceaction = 0 THEN err($"Contents of This Entity may
        not be changed");                                            01246
        IF replaceaction = defaultreplace THEN                       01247
           IF macromode THEN replrecord(&rd)                         01248
           ELSE replfield(&rd)                                       01249
        ELSE                                                         01250
           BEGIN                                                     01251
           callreplaceproc(&rd);                                     01252
           END;                                                      01253
        IF dynamicrecord THEN                                        01254
           BEGIN                                                     01255
           initialised _ FALSE;                                      01256
           initrd(&rd);                                              01257
           END;                                                      01258
        RETURN;                                                      01259
        END.                                                         01260
     (setfield)PROC(rd, value);                                      01261
        %Accepts a rdecord descripton, and a value.  Sets the
        defined field to the value%                                  01262
        LOCAL fld, bp;                                               01263
        REF rd, fld;                                                 01264
        IF NOT initialised THEN err($"Illegal Replace");             01265
        setfieldptr;                                                 01266
        bp _ (fieldptr + recordstart) .A noindexmask;                01267
        .bp _ value;                                                 01268
        RETURN;                                                      01269
        END.                                                         01270
  (sequence)PROC(rd, symtype, symval);                               01271
     %Standard parms...Parse Sequence%                               01272
     REF rd;                                                         01273
     copyrd(symval+1, &rd);                                          01274
     RETURN;                                                         01275
     END.                                                            01276
  (showaddr)PROC(rd);                                                01277
     %Parms are:                                                     01278
        rd = address of rd                                           01279
     Returns updated fd%                                             01280
     LOCAL symtype, symval;                                          01281
     REF rd;                                                         01282
     echon();                                                        01283
     IF NOT getsym( :symtype, symval) THEN parsevalue(&rd, 0, 0)
                                                                     01284
```

```
      ELSE IF stacksymbol THEN stackref(&rd, symtype, symval)   01285
           ELSE IF sequencename THEN sequence(&rd, symtype, symval)
                                                                01286
                ELSE parsevalue(&rd, symtype, symval);          01287
      IF NOT initialised THEN initrd(&rd);                      01288
      RETURN;                                                   01289
      END.                                                      01290
%* BREAKPOINT Routines%                                        01291
   (bpclall)PROC;                                               01292
      %Clear all breakpoints%                                   01293
      LOCAL bpn, bpa;                                           01294
      bpn _ -1;                                                 01295
      WHILE (bpn _ bpn+1) < nbkpts DO                           01296
         IF (bpa _ findbp(bpn)) THEN                            01297
            BEGIN                                               01298
            bpclr(bpa);                                         01299
            END;                                                01300
      RETURN;                                                   01301
      END.                                                      01302
   (bpclr)PROC(bpa);                                            01303
      LOCAL c;                                                  01304
      %Arg is address of bp%                                    01305
      IF tblsearch($lbptbl, lbptblsz, lbptblentsz, bpa: c) THEN [c] _
      0; %Clear local table entry%                              01306
      [bpa] _ 0;                                                01307
      RETURN;                                                   01308
      END.                                                      01309
   (bppntall)PROC;                                              01310
      %Print all breakpoints%                                   01311
      LOCAL bpn;                                                01312
      bpn _ -1;                                                 01313
      WHILE (bpn _ bpn+1) < nbkpts DO                           01314
         IF findbp(bpn: bpn) THEN typbrk(bpn);                  01315
      RETURN;                                                   01316
      END.                                                      01317
   (breakpoint)PROC;                                            01318
      %Get hee to check whether or not  o execute a break...calld
      with a !JSP brkchk%                                       01319
      (brkchk):                                                 01320
         !HRRZM 0,ddtrloc; %Save off return address%            01321
         !JSP ddtsvrg; %Save off regusters%                     01322
         IF (ddtrp _ findbp(ddtrloc-1)) = 0 THEN                01323
            BEGIN                                               01324
            crlf();                                             01325
            typeas($"Unrecognised Break Point");                01326
            !JSP ddtresreg;                                     01327
            !JRST @ddtrloc;                                     01328
            END;                                                01329
         CASE [ddtrp].bptest OF                                 01330
            =0: NULL;                                           01331
            =1: %Trace%                                         01332
               BEGIN                                            01333
               typtrace(ddtrloc-1);                             01334
               GOTO bkpret;                                     01335
               END;                                             01336
            < 1B7: %Procedure Call..T/F%                        01337
```

```
            BEGIN                                                   01338
            ddtinst _ [ddtrp].bptest; %Address of proc to be
            called%                                                 01339
            !JSP ddtr1reg; %restore registers except 0 and s%
                                                                    01340
            !CALL0 @ddtinst; %ll Routine%                           01341
            IF NOT a1 THEN GOTO bkpret;                             01342
            END; %Fall Through to break%                            01343
         ENDCASE  %Fall through to break%                           01344
            BEGIN                                                   01345
            ddtjunk _ [ddtrp].bpval;                                01346
            ddtinst _ [ddtrp].bptest; %Instruction%                 01347
            !JSP ddtr1reg; %Restore registers except s and 0%
                                                                    01348
            r0 _ ddtjunk;                                           01349
            !XCT ddtinst;                                           01350
            GOTO bkpret; %Test Failed%                              01351
            END; %Fall through to break%                            01352
      %By here, we will execute the break point%                    01353
         %Resyore registers%                                        01354
            !JSP ddtresreg;                                         01355
         %Now call ddt, with breakpoint address on top of stack%
                                                                    01356
            !PUSH s,ddtrloc; %Return address in case breakpoint
            cleared%                                                01357
            !JSP breakddt; %Assumes top of stack contains break
            address...returns with break address in 3Z(s)%         01358
      %Returning from ddt....proceed%                               01359
         (breakret): %Returning from breakpon execution%            01360
         %Restore return address at 3(s)%                           01361
            !MOVE 3(s);                                             01362
            !MOVEM ddtrloc;                                         01363
         %Save registers%                                           01364
            !JSP ddtsvrg;                                           01365
         %See if breakpoit is still valid%                          01366
            IF (ddtrp _ findbp(ddtrloc-1)) = 0 THEN %No More Brek
            Point%                                                  01367
               %No break any more...restore registers and return%
                                                                    01368
               BEGIN                                                01369
               ddtrloc _ ddtrloc-1;                                 01370
               !JSP ddtresreg;                                      01371
               !JRST @ddtrloc;                                      01372
               END;                                                 01373
      %Global breakpoint return code%                               01374
         (bkpret):                                                  01375
            %Breakpoint is still there...execute instruction and
            proceed%                                                01376
               ddtinst _ [ddtrp].bpinst;                            01377
               !JSP ddtresreg;                                      01378
               !XCT ddtinst;                                        01379
               !JRST @ddtrloc;                                      01380
   %Miscellaneous breakpoint routines%                              01381
      (ddtsvrg): %Save registers on stack%                          01382
         ddtrg1 _ r0;                                               01383
         ddtrg2 _ r1;                                               01384
```

```
            !MOVEI 1,2(s); %Location to save registers%                    01385
            !JSP ddtsvr1; %Save registers 2-17%                           01386
            !PUSH s,ddtrg2; %Place original r1 into proper place%
                                                                          01387
                                                                          01388
            !HRRI 1,16B;                                                  01389
            !HRL 1,1;                                                     01390
            !ADD s,1; %Fix up s%                                          01391
            !JRST @ddtrg1; %RETURN%                                       01392
         (ddtresreg): %Restore the registers%                             01393
            ddtrg1 _ r0;                                                  01394
            r1.LH _ s.RH-15B; %Location of registers%                     01395
            r1.RH _ 2; %Where they are to go%                             01396
            r0 _ 17B; %Last loc%                                          01397
            !BLT 1,@0;                                                    01398
            !MOVE 1,1(s); %Restore 1%                                     01399
            !JRST @ddtrg1; %RETURN%                                       01400
         (ddtr1reg): %Restore the registers except for s%                 01401
            ddtrg1 _ r0;                                                  01402
            r1.LH _ s.RH-15B; %Location of registers%                     01403
            r1.RH _ 2; %Where they are to go%                             01404
            r0 _ 17B; %Last loc%                                          01405
            !MOVEM s,ddtrg2 ; %Save off s%                                01406
            !BLT 1,@0;                                                    01407
            !MOVE 1,1(s); %Restore 1%                                     01408
            !MOVE s,ddtrg2;                                               01409
            !JRST @ddtrg1;                                                01410
      END.                                                                01411
   (chkbp)PROC(loc);
      %This procedure checks the bkpt num/ addr passed for first a
      stack manipulation instruction, and secondly an already
      defined breakpoint.                                                 01412
      Returns addr updated for stack manip and breakpoints as first
      result,                                                             01413
      If previously defined breakpoint, returns breakpoint address +
      number%                                                             01414
      LOCAL bpa, bpn;                                                     01415
      bpa _ bpn _ 0;                                                      01416
      IF (loc > nbkpts) AND [loc] .A smanipumask = smanipulator THEN
                                                                          01417
         BEGIN                                                            01418
         BUMP loc;                                                        01419
         END;                                                             01420
      IF (bpa _ findbp(loc: bpn)) THEN                                    01421
         BEGIN %Brekpoint already there%                                  01422
         loc _ [bpa];                                                     01423
         END;                                                             01424
      RETURN(loc, bpa, bpn);                                              01425
      END.                                                                01426
   (restbp)PROC;                                                          01439
      %Restore the original instructions%                                 01440
      LOCAL c, bptinst;                                                   01441
      bptinst _ jsp + $brkchk;                                            01442
      c _ -bptblentsz;                                                    01443
      WHILE (c _ c+bptblentsz) < bptblsz DO                               01444
         IF bptbl[c] # 0 AND [bptbl[c]] = bptinst THEN                    01445
            [bptbl[c]] _ bptbl[c].bpinst;                                 01446
```

```
    RETURN;                                                          01447
    END.                                                             01448
(setbkpt)PROC(addr, test, testval, bpa, bpn);                        01449
    %Create a breakpoint%                                            01450
    %Return brekpoint number%                                        01451
    LOCAL lbpa; %Brealpoint address, number%                         01452
    IF NOT bpa THEN %Get slot in table%                              01453
       BEGIN                                                         01454
       IF NOT tblsearch($lbptbl, lbptblsz, 1, 0: lbpa) THEN
       err($"Breakpoint Table Full");                                01455
       IF NOT tblsearch($bptbl, bptblsz, bptblentsz, 0: bpa, bpn)
       THEN err($"Breakpoint Table Full");                           01456
       [lbpa] _ bpa;                                                 01457
       END;                                                          01458
    IF [addr] .A smanipumask = smanipulator THEN                     01459
       BUMP addr;                                                    01460
    [bpa].bpaddr _ addr;                                             01461
    [bpa].bptest _ test;                                             01462
    [bpa].bpval _ testval;                                           01463
    RETURN(bpn);                                                     01464
    END.                                                             01465
(storbp)PROC;                                                        01466
    %Store the breakpoints in the table%                             01467
    LOCAL c, bptinst;                                                01468
    bptinst _ jsp + $brkchk;                                         01469
    c _ -bptblentsz;                                                 01470
    WHILE (c _ c+bptblentsz) < bptblsz DO                            01471
       IF bptbl[c] # 0 AND [bptbl[c]] # bptinst THEN                 01472
          BEGIN                                                      01473
          bptbl[c].bpinst _ [bptbl[c]]; %Save off instruction%
                                                                     01474
          [bptbl[c]] _ bptinst;                                      01475
          END;                                                       01476
    RETURN;                                                          01477
    END.                                                             01478
(typbrk)PROC(bp);                                                    01479
    LOCAL bpa, bpn;                                                  01480
    LOCAL STRING tempsr[50];                                         01481
    IF NOT (bpa _ findbp(bp: bpn)) THEN err($"No Such Breakpoint");
                                                                     01482
    *tempsr* _ "BP #", STRING(bpn), ", At ";                         01483
    crlf();                                                          02768
    typeas($tempsr);                                                 01484
    typval([bpa].bpaddr, symbolicmode);                              01485
    IF [bpa].bptest # 0 THEN                                         01486
       BEGIN                                                         01487
        CASE [bpa].bptest OF                                         01488
          =1:                                                        01489
             *tempsr* _ "(Trace)";                                   01490
          <1B7:                                                      01491
             BEGIN                                                   01492
             ddgtsymbol($tempsr, [bpa].bptest);                      01493
             *tempsr* _ "Test Procedure is ", *tempsr*;              01494
             END;                                                    01495
          ENDCASE                                                    01496
             BEGIN                                                   01497
```

```
            ddgtsymbol($tempsr, [bpa].bptest.RH);                    01498
            *tempsr* _ *tempsr*, SP;                                 01499
            CASE [bpa].bptest .A opcodmask OF                        01500
               =came: *tempsr* _ *tempsr*, "=;                       01501
               =camn: *tempsr* _ *tempsr*, "#;                       01502
               =caml: *tempsr* _ *tempsr*, ">;                       01503
               =camg: *tempsr* _ *tempsr*, "<;                       01504
               =camle: *tempsr* _ *tempsr*, ">=";                    01505
               =camge: *tempsr* _ *tempsr*, "<=";                    01506
               ENDCASE err($"Illegal Breakpoint Relation");          01507
            *tempsr* _ "Test: ",*tempsr*, SP,
            *[octnum([bpa].bpval)]*;                                 01508
            END;                                                     01509
         crlf();                                                     01510
         typeas($tempsr);                                            01511
         END;                                                        01512
      crlf();                                                        01513
      RETURN;                                                        01514
      END.                                                           01515
   (typtrace)PROC(addr);                                             01516
      %Types out a trace message for address%                        01517
      LOCAL STRING tempsr[50];                                       01518
      ddgtsymbol($tempsr, addr); %Get a symbol for it%               01519
      %Use TENEX output stuff because crlf or typeas may be traced%
                                                                     01520
         !pbout(CR); !pbout(LF);                                     01521
         !sout (101B, chbmty + $tempsr, -tempsr.L);                  01522
      RETURN;                                                        01523
      END.                                                           01524
   (xbpclear)PROC;                                                   01525
      %Clear a breakpoint%                                           01526
      LOCAL bpa, bpn;                                                01527
      echo($"Clear ");                                               01528
      echon();                                                       01529
      CASE nextchar OF                                               01530
         ='A:                                                        01531
            BEGIN                                                    01532
            echo($"ll");                                             01533
            chkcacr();                                               01534
            bpclall();                                               01535
            END;                                                     01536
         ENDCASE                                                     01537
            BEGIN                                                    01538
            backupchar;                                              01539
            bpa _ parsevalue(0, 0, 0);                               01540
            chkcacr();                                               01541
            IF [bpa] .A smanipumask = smanipulator THEN              01542
               BUMP bpa;                                             01543
            IF NOT (bpa _findbp(bpa: bpn)) THEN err($"No Such
            Breakpoint");                                            01544
            bpclr(bpa);                                              01545
            bpclr(bpn);                                              01546
            END;                                                     01547
      RETURN;                                                        01548
      END.                                                           01549
   (xbpprint)PROC;                                                   01550
```

```
    %Print breakpoint%                                                  01551
    LOCAL bpa, bpn;                                                     01552
    echo($"Print ");                                                    01553
    echon();                                                            01554
    CASE nextchar OF                                                    01555
      ='A:                                                              01556
        BEGIN                                                           01557
        echo($"11");                                                    01558
        chkcacr();                                                      01559
        bppntall();                                                     01560
        END;                                                            01561
      =CA:                                                              01562
        bppntall();                                                     01563
      ENDCASE                                                           01564
        BEGIN                                                           01565
        backupchar;                                                     01566
        bpa _ parsevalue(0, 0, 0);                                      01567
        chkcacr();                                                      01568
        typbrk(bpa);                                                    01569
        END;                                                            01570
    RETURN;                                                             01571
    END.                                                                01572
  (xbpset)PROC;                                                         01573
    %This is the procedure for setting breakpoints%                     01574
    LOCAL bpa, bpn, loc, testval, test;                                 01575
    echo($"Set at ");                                                   01576
    echon();                                                            01577
    loc _ parsevalue(0, 0, 0);                                          01578
    loc _ chkbp(loc: bpa, bpn);                                         01579
    test _ testval _ 0;                                                 01580
    LOOP                                                                01581
      BEGIN                                                             01582
      echoff();                                                         01583
      CASE nextchar OF                                                  01584
        =C.: NULL;                                                      01585
        =CA, =CR: EXIT LOOP;                                            01586
        ='C:                                                            01587
          BEGIN                                                         01588
          echo($"Call ");                                               01589
          echon();                                                      01590
          IF  NOT (test _ gprcname()) THEN err($"Not a
          Procedure");                                                  01591
          END;                                                          01592
        ='R:                                                            01593
          BEGIN                                                         01594
          echo($"Replaces Breakpoint: ");                               01595
          echon();                                                      01596
          IF NOT (bpa _ findbp(parsevalue(0, 0, 0): bpn)) THEN
          err($"No Such Breakpoint");                                   01597
          END;                                                          01598
        ='T:                                                            01599
          BEGIN                                                         01600
          echo($"Test (Location relation value): ");                    01601
          echon();                                                      01602
          test _ parsevalue(0, 0, 0) .A 18M;                            01603
          deblnk();                                                     01604
```

```
                CASE nextchar OF                                                01605
                   ='=: test _ test .V came;                                    01606
                   ='#: test _ test .V camn;                                    01607
                   ='<:                                                         01608
                      IF nextchar = '= THEN test _ test .V camge                01609
                      ELSE                                                      01610
                         BEGIN                                                  01611
                         backupchar;                                            01612
                         test _ test .V camg;                                   01613
                         END;                                                   01614
                   ='>:                                                         01615
                      IF nextchar = '= THEN test _ test .V camle                01616
                      ELSE                                                      01617
                         BEGIN                                                  01618
                         backupchar;                                            01619
                         test _ test .V caml;                                   01620
                         END;                                                   01621
                    =CA, =CR: EXIT LOOP;                                        01622
                    ENDCASE err($"Illegal Test");                               01623
                 deblnk();                                                      01624
                 testval _ parsevalue(0, 0, 0);                                 01625
                 END;                                                           01626
              ENDCASE err($"??");                                               01627
           END;                                                                 01628
        bpn _ setbkpt(loc, test, testval, bpa, bpn);                            01629
        RETURN;                                                                 01630
        END.                                                                    01631
     (xbrkpt)PROC;                                                              01632
        %This the breakpoint control routine%                                   01633
        echo($"Breakpoint ");                                                   01634
        CASE nextchar OF                                                        01635
           ='C: xbpclear();                                                     01636
           ='P: xbpprint();                                                     01637
           ='S: xbpset();                                                       01638
           ENDCASE                                                              01639
              BEGIN                                                             01640
              echo($"? (Clear/Print/Set) ");                                    01641
              REPEAT CASE;                                                      01642
              END;                                                              01643
        RETURN;                                                                 01644
        END.                                                                    01645
  %* CONTINUE command %                                                        01646
     (xcontin)PROC;                                                             01647
        echo($"Continue");                                                      01648
        chkcacr();                                                              01649
        crlf();                                                                 01650
        storbp();                                                               01651
        SIGNAL(continsig);                                                      01652
        END.                                                                    01653
  %* DEFINE Command %                                                          01654
     (xdddef)PROC;                                                              01655
        % this routine parses and processes the DEFINE symbol command %       01656

        % a symbol is entered into the ddtsymbol table with the
        specified value only if it does not already exist there (NEW)
        or only if it already exists there (OLD) %                              01657
```

```
      LOCAL value, addrv, char;                                          01658
      LOCAL STRING tempstr[20];                                          01659
      echo($"DEFINE ");                                                  01660
      CASE char _ nextchar OF                                            01661
         ='N: % NEW symbol %                                             01662
            BEGIN                                                        01663
            echo($"New Symbol: ");                                       01664
            IF ddmrkx = 0                                                01665
               THEN err($"No Block in Mark Stack for Definition");
                                                                         01666
            echon();                                                     01667
            IF NOT rdsym($tempstr) THEN err($"Illegal Symbol");          01668
               chkcacr();                                                01669
            IF ddtlookup($tempstr, 0 : value,addrv)                      01670
               THEN err($"Symbol Already Exists");                       01671
            echo($" New value = ");                                      01672
            IF NOT parsenter( :value) THEN err($"Illegal Value");
                                                                         01673
               chkcacr();                                                01674
            ddtenter( $tempstr, value, global );                         01675
            END;                                                         01676
         ='O: % OLD symbol %                                             01677
            BEGIN                                                        01678
            echo($"OLD Symbol: ");                                       01679
            echon();                                                     01680
            IF NOT rdsym($tempstr) THEN err($"Illegal Symbol");          01681
               chkcacr();                                                01682
            IF NOT ddtlookup($tempstr, 0 : value,addrv)                  01683
               THEN err($"Symbol Does Not Exist");                       01684
            echo($" New value = ");                                      01685
            IF NOT parsenter( :value) THEN err($"Illegal Value");
                                                                         01686
               chkcacr();                                                01687
            [addrv] _ value; % stick in new value %                      01688
            END;                                                         01689
      ENDCASE                                                            01690
         BEGIN                                                           01691
         echo($"? (New/Old) ");                                          01692
         REPEAT CASE;                                                    01693
         END;                                                            01694
      RETURN;                                                            01695
      END.                                                               01696
%* FIND %                                                               01697
   (xddfind) PROCEDURE;                                                  01698
      % this routine parses and processes the FIND command %             01699
      % this command finds all occurrences of a particular value
      between two specified address limits and prints out the
      addresses of the hits %                                            01700
      LOCAL lower,upper, default;                                        01701
      LOCAL value,mask;                                                  01702
      LOCAL STRING tempstr[75];                                          01703
      LOCAL STRING tempstr1[50];                                         01704
      echon();                                                           01705
      echo($"Find Content ");                                            01706
      IF NOT parsenter( :value) THEN err($"Invalid Content
      Expression");                                                      01707
```

```
                                                                    01708
      chkcacr();                                                    01709
      echo($" Masked by ");
      IF NOT parsenter( :mask) THEN mask _ 777777B; % defalut mask %;
                                                                    01710
                                                                    01711
      chkcacr();                                                    01712
      echo($" From: ");                                             01713
         lower _ parsevalue(0,0,0 : default);                       01714
         chkcacr();                                                 01715
         IF default THEN lower _ $sysrtn;                           01716
      echo($" To: ");                                               01717
         upper _ parsevalue(0,0,0 : default);                       01718
         chkcacr();                                                 01719
         IF default THEN lower _ 554000B;                           01720
      crlf();                                                       01721
      FOR lower UP UNTIL > upper DO                                 01722
         BEGIN                                                      01723
         IF value = [lower] .A mask THEN                            01724
            BEGIN % found a match  -- now print it out %
            ddgtsymbol( $tempstr, lower); % get representation for
                                                                    01725
            address %
            ddgtsymbol( $tempstr1, [lower]); % get representation for
                                                                    01726
            value %                                                 01727
            % edit them together %                                 01728
            *tempstr* _ *tempstr*, "/  ", *tempstr1*;               01729
            typeas($tempstr);                                       02767
            crlf();                                                 01730
            END;                                                    01731
         IF inpstp THEN                                             01732
            BEGIN                                                   01733
            inpstp _ FALSE;                                         01734
            EXIT LOOP;                                              01735
            END;                                                    01736
         END;                                                       01737
      RETURN;                                                       01738
      END.                                                          01739
%* GO TO Command %                                                 01740
   (xgo)PROC;                                                       01741
      LOCAL v;                                                      01742
      echo($"Go to Location: ");                                    01743
      echon();
      IF (v _ parsevalue(0, 0, 0)) < 1000 THEN err($"Illegal
                                                                    01744
      Address");                                                    01745
      chkcacr();                                                    01746
      storbp();                                                     01747
      SIGNAL(gosig, v); %To be trapped by runddt%                   01748
      END.                                                          01749
%* MARK command %                                                  01750
   (xnmark)PROC;                                                    01751
      %This the mark symbol table control routine%                  01752
      echo($"Mark Symbol Table: ");                                 01753
      CASE nextchar OF                                              01754
         ='C: xddpop();                                             01755
         ='P: xmkprint();                                           01756
         ='S: xddmark();                                            01757
         ENDCASE                                                    01758
            BEGIN
```

```
            echo($"? (Clear/Print/Set) ");                                  01759
            REPEAT CASE;                                                    01760
            END;                                                            01761
      RETURN;                                                               01762
      END.                                                                  01763
   (xmkprint) PROCEDURE;                                                    01764
      % this routine parses and processes the MARK SYMBOL TABLE PRINT
      command %                                                             01765
      LOCAL i,j,k;                                                          01766
      LOCAL STRING tempstr[50];                                             01767
      echo($"Print");                                                       01768
      chkcacr();                                                            01769
      crlf();                                                               01770
      IF ddmrkx THEN *tempstr* _ "Contents of Mark Stack:"                  01771
      ELSE *tempstr* _  "Mark Stack Empty";                                 01772
      typeas( $tempstr );                                                   01773
      FOR i _ ddmrkx - 1 DOWN UNTIL < 0 DO                                  01774
         BEGIN                                                              01775
         j _ [ddmrk[i]-1]; % RADIX50 name representation %                  01776
         *tempstr* _ NULL;                                                  01777
         crlf();                                                            01778
         LOOP % convert from radix 50 to ascii %                            01779
            BEGIN                                                           01780
            DIV j/50B, j, k;                                                01781
            IF NOT k THEN EXIT LOOP;                                        01782
            k _ IF (k _ k + 47) > 57 THEN k+7 ELSE k;                       01783
            *tempstr* _ k , *tempstr*;                                      01784
            END;                                                            01785
         typeas($tempstr);                                                  01786
         END;                                                               01787
      RETURN;                                                               01788
      END.                                                                  01789
   (xddmark) PROCEDURE;                                                     01790
      % this routine parses and processes the MARK SYMBOL TABLE
      command %                                                             01791
      LOCAL i,j,k;                                                          01792
      LOCAL STRING tempstr[50];                                             01793
      echo($"Set At: ");                                                    01794
      echon();                                                              01795
      IF NOT rdsym($tempstr) THEN err($"Illegal Block Name");               01796
      chkcacr();                                                            01797
      ddtmark($tempstr);                                                    01798
      RETURN;                                                               01799
      END.                                                                  01800
   (xddpop) PROCEDURE;                                                      01801
      % this routine parses and processes the POP SYMBOL TABLE
      command %                                                             01802
      LOCAL i,j,blockptr,namevalue;                                         01803
      LOCAL STRING tempstr[50];                                             01804
      echo($"Clear Block ");                                                01805
      echon();                                                              01806
      IF NOT rdsym($tempstr) THEN err($"Illegal Block Name");               01807
      chkcacr();                                                            01808
      namevalue _ ddtname( $tempstr, 1) .A 32M;                             01809
      % try to find the names symbol in the mark stack %                   01810
      FOR i _ ddmrkx - 1 DOWN UNTIL < 0 DO                                  01811
```

```
        BEGIN                                                          01812
        blockptr _ ddmrk[i];                                           01813
        IF [blockptr-1] = namevalue THEN                               01814
           BEGIN % found block, now collapse mark stack %              01815
           FOR j _ i UP UNTIL >= ddmrkx - 1  DO                        01816
              ddmrk[j] _ ddmrk[j+1];                                   01817
           ddmrkx _ ddmrkx - 1;                                        01818
           % check to see if block being deleted is at the bottom of
           the DDT symbol table %                                      01819
           ddtchkp( blockptr );                                        01820
           RETURN;                                                     01821
           END;                                                        01822
        END;                                                           01823
      err($"Block Not In Mark Stack");                                 01824
      END.                                                             01825
%* PROCEDURE call, replace, backup%                                    01826
   (gprcname)PROC;                                                     01827
      LOCAL symval, symtype, c;                                        01828
      IF NOT getsym(: symtype, symval) THEN RETURN(0);                 01829
      IF ([symval].accum10 # $s AND [symval].addr10 # $sysovr) THEN
                                                                       01830
         IF [symval].opcod10 # 254B %JRST% THEN RETURN(0)              01831
         ELSE                                                          01832
            IF tblsearch($proctbl, proctblsz, proctblentsz, symval:
            c) THEN RETURN(symval, c)                                  01833
            ELSE RETURN(FALSE);                                        01834
      RETURN(symval, 0);                                               01835
      END.                                                             01836
   (xbackp)PROC;                                                       01837
      LOCAL pr1, c;                                                    01838
      echo($"Back up to ");                                            01839
      echon();                                                         01840
      IF NOT (pr1 _ gprcname( :c)) THEN err($"Illegal Procedure
      Name");                                                          01841
      chkcacr();                                                       01842
      IF c = 0 THEN err($"Not in Replace List");                       01843
      [pr1] _ [c+1]; %Original inst%                                   01844
      [c] _ 0; %Clear it%                                              01845
      RETURN;                                                          01846
      END.                                                             01847
   (xcallp)PROC(prcad, frm, fp);                                       01848
      %Fake a call to prcad%                                           01849
      storbp(); %Set up breakpoints%                                   01850
      ddtjunk _ prcad;                                                 01851
      s _ m;                                                           01852
      IF fp > 0 THEN                                                   01853
         BEGIN                                                         01854
         r2.LH _ frm;                                                  01855
         r2.RH _ m+1;                                                  01856
         r3 _ m.RH + fp;                                               01857
         !BLT 2,@3;                                                    01858
         END;                                                          01859
      !JSP urstrg;                                                     01860
      GOTO [ddtjunk];                                                  01861
      END.                                                             01862
   (xcalprc)PROC;                                                      01863
```

```
    LOCAL prcadr, frm[maxprms], fp, char, rd, c, symtype, symval;
    REF  rd;
    echo($"Call ");
    echon();
    IF NOT prcadr _ gprcname() THEN err($"Not a Procedure");
    fp _ 0;
    IF (char _ nextchar) = '( THEN
       LOOP
          BEGIN
          parsenter(:frm[fp]);
          IF (fp _ fp+1) > maxprms THEN err($"Too Many
          Parameters");
          IF (char _ nextchar) # ', AND char # '/ THEN
             IF char # ') THEN err($"Illegal Paremeter")
             ELSE EXIT;
          END
    ELSE
       IF char # CA THEN
          BEGIN
          IF NOT getsym(: symtype, symval)
          OR NOT stacksymbol THEN err($"Illegal Frame Designator");
          &rd _ symval+1;
          framep _ CASE recordstart OF
             =frametoptype: frametop;
             ENDCASE framep;
          initframe();
          IF nparms > maxprms THEN err($"Too Many Parameters");
          c _ 0;
          WHILE (c_c+1) <= nparms DO
             frm[fp := fp+1] _ [framep + c];
          END
       ELSE backupchar;
    chkcacr();
    %Clear input buffer so we can read characters%
       clrbuf(0);
    crlf();
    xcallp(prcadr, $frm, fp);
    c _ a1; %save off result%
    !JSP usvreg; %Save off user registers%
    restbp(); %Restore break points%
    clrbuf(0); %Get rid of any junk left over%
    crlf();
    typeas( $"Return Value = ");
    typval(c, numericmode); %Type out result%
    RETURN;
    END.
 (xprocc)PROC;
    echo($"Procedure ");
    CASE nextchar OF
       ='C:
          xcalprc();
       ='R:
          xreplp();
```

```
                                                                    01916
        ="B:                                                        01917
           xbackp();                                                01918
        ENDCASE                                                     01919
           BEGIN                                                    01920
           echo($"? (Backup/Call/Replace) ");                       01921
           REPEAT CASE;                                             01922
           END;                                                     01923
     RETURN;                                                        01924
     END.                                                           01925
  (xreplp)PROC;                                                     01926
     LOCAL pr1, pr2, c;                                             01927
     echo($"Replace Procedure ");                                   01928
     echon();
     IF NOT (pr1 _ gprcname()) THEN err($"Illegal Procedure Name");
                                                                    01929
                                                                    01930
     chkcacr();                                                     01931
     echo($" By ");
     IF NOT (pr2 _ gprcname()) THEN err($"Illegal Procedure Name");
                                                                    01932
                                                                    01933
     chkcacr();
     IF NOT tblsearch($proctbl, proctblsz, proctblentsz, 0: c) THEN
                                                                    01934
                                                                    01935
        BEGIN                                                       01936
        crlf();
        typeas($"Backup Table Full...Replaced Instruction is:");
                                                                    01937
                                                                    01938
        typval([pr1], symbolicmode);                                01939
        END                                                         01940
     ELSE                                                           01941
        BEGIN                                                       01942
        [c] _ pr1;                                                  01943
        [ c +1] _ [ pr1];                                           01944
        END;                                                        01945
     [pr1] _ jrst + pr2;                                            01946
     RETURN;                                                        01947
     END.                                                           01948
%* RECORD Pointer Set%                                             01949
  (xrpset)PROC;                                                     01950
     %Set up the record pointer RECP%                               01951
     LOCAL symval, symtype;                                         01952
     echo($"Record Pointer Set to: ");                              01953
     echon();                                                       01954
     recprep1(0);                                                   01955
     RETURN;                                                        01956
     END.                                                           01957
%* SHOW%                                                          01958
  (chkmode)PROC( caflag);                                           01959
     LOCAL char, mode;                                              01960
     mode _ currenttypemode;                                        01961
     echoff();                                                      01962
     IF NOT caflag THEN RETURN(mode);                               01963
     IF (char _ nextchar) = C. THEN                                 01964
        BEGIN                                                       01965
        CASE nextchar OF                                            01966
           ="S:                                                     01967
              BEGIN
```

```
                echo($" Symbolic  ");                                   01968
                mode _ symbolicmode;                                    01969
                END;                                                    01970
             ="N:                                                       01971
                BEGIN                                                   01972
                echo($" Numeric  ");                                    01973
                mode _ numericmode;                                     01974
                END;                                                    01975
             ="T:                                                       01976
                BEGIN                                                   01977
                echo($" Text  ");                                       01978
                mode _ textmode;                                        01979
                END;                                                    01980
             ENDCASE err($"Illegal Mode Spec");                         01981
          END                                                           01982
       ELSE backupchar;                                                 01983
       IF char = "[ THEN mode _ numericmode;                            01984
       IF char = "/ THEN mode _ symbolicmode;                           01985
       currenttypemode _ mode;                                          01986
       RETURN(mode);                                                    01987
       END.                                                             01988
    (xnshow)PROC(oldrd, echflg);                                        01989
       LOCAL rd[ribsize], mode, caflag, char;                           01990
       REF oldrd;                                                       01991
       IF echflg THEN echo($"Show ");                                   01992
       copyrd(&oldrd, $rd);                                             01993
       caflag _ TRUE;                                                   01994
       CASE (char _ nextchar) OF                                        01995
          ="R: %Record%                                                 01996
             BEGIN                                                      01997
             echo($"Record ");                                          01998
             IF recp = 0 THEN err($"Record Pointer Undefined");         01999
             IF NOT recordentity OR recordname # recp THEN              02000
                BEGIN                                                   02001
                entity _ recordtype;                                    02002
                dynamicrecord _ FALSE;                                  02003
                recordname _ recp;                                      02004
                initialised _ FALSE;                                    02005
                END;                                                    02006
             addressmode _ macro;                                       02007
             END;                                                       02008
          ="S: %String%                                                 02009
             BEGIN                                                      02010
             echo($"String ");                                          02011
             entity _ stringtype;                                       02012
             dynamicrecord _ TRUE;                                      02013
             defineproc _ $strngrec;                                    02014
             addressmode _ macro;                                       02015
             initialised _ FALSE;                                       02016
             END;                                                       02017
          ="L:                                                          02018
             BEGIN                                                      02019
             echo($"Location ");                                        02020
             REPEAT(char _ SP);                                         02021
             END;                                                       02022
          ="^:                                                          02023
```

```
        %Predecessor%                                                   02024
        BEGIN                                                           02025
        IF echflg THEN echo($"Preceeding ");                            02026
        chngadr($rd, -1);                                               02027
        (reltype): %Come here to type relative%                         02028
           typloc($rd); %Type out address%                              02029
           caflag _ 0;                                                  02030
           GOTO comtype;                                                02031
        END;                                                            02032
     =LF, ="N:                                                          02033
        %Next%                                                          02034
        BEGIN                                                           02035
        IF echflg THEN echo($"Next ");                                  02036
        chngadr($rd, 1);                                                02037
        GOTO reltype;                                                   02038
        END;                                                            02039
     =CA: % command accept -- may be a bug %                            02040
        BEGIN                                                           02041
        backupchar;                                                     02042
        IF displayflag THEN REPEAT CASE (SP) ELSE GOTO comtype;         02043
                                                                        02044
        END;                                                            02045
     ="_,  % assign a value %                                           02046
     =CR, =C.: %Same thing over%                                        02047
        BEGIN                                                           02048
        backupchar;                                                     02049
        GOTO comtype;                                                   02050
        END;                                                            02051
     =TAB: %Move address to last value%                                 02052
        BEGIN                                                           02053
        recordstart _ lv;                                               02054
        caflag _ 0;                                                     02055
        GOTO comtype;                                                   02056
        END;                                                            02057
     ENDCASE                                                            02058
        BEGIN                                                           02059
        %Set to location / word%                                        02060
        IF char # SP THEN                                               02061
           BEGIN                                                        02062
           typech(char);                                                02063
           backupchar;                                                  02064
           END;                                                         02065
        entity _ wordtype;                                              02066
        dynamicrecord _ TRUE;                                           02067
        defineproc _ $wordrec;                                          02068
        mother _ 0;                                                     02069
        addressmode _ macro;                                            02070
        initialised _ FALSE;                                            02071
        END;                                                            02072
     %Now read and parse the address%                                   02073
      showaddr($rd);                                                    02074
    %Now type it out%                                                   02075
      (comtype):                                                        02076
      %Check for mode%                                                  02077
         mode _ chkmode(caflag);                                        02078
      typentity($rd, mode);
```

```
        %Now update old address%                                        02079
           copyrd($rd, &oldrd);                                         02080
        %Replace if appropriate%                                        02081
           IF caflag THEN repval(&oldrd);                               02082
        RETURN;                                                         02083
        END.                                                            02084
    %* TRACE%                                                          02085
        (xtrace)PROC;                                                   02086
           LOCAL taddr, bpn, bpa;                                       02087
           %Set up trace%                                               02088
           echo($"Trace Location ");                                    02089
           echon();                                                     02090
           IF (taddr _ chkbp(parsevalue(0, 0, 0): bpa, bpn)) < 1000 THEN
           err($"??");                                                  02091
           chkcacr();                                                   02092
           setbkpt(taddr, 1, 0, bpa, bpn);                              02093
           RETURN;                                                      02094
           END.                                                         02095
    %* VALUE Command%                                                  02096
        (xtval)PROC;                                                    02097
           LOCAL v;                                                     02098
           echo($"Value of ");                                          02099
           echon();                                                     02100
           v _ parsevalue(0, 0, 0);                                     02101
           typeas($" is ");                                             02102
           typval(v, chkmode(TRUE));                                    02103
           chkcacr();                                                   02104
           RETURN;                                                      02105
           END.                                                         02106
%* Record Information Block Routines%                                  02107
    %* RIB Utilities%                                                  02108
        (initrd)PROC(rd);                                               02109
           %Initialises an RD...Assumes that the followinig fields are
           valid when called:                                           02110
              recordstart                                               02111
              mother                                                    02112
              recordname                                                02113
              dynamicrecord (and defineproc if applicable)              02114
              entity                                                    02115
              addressmode                                               02116
           Initialises the following filds to 0 in all cases:           02117
              fieldnumber                                               02118
              daughter                                                  02119
              suppressloc                                               02120
           Calls the defineproc if dynamicrecord, or otherwise setsup
                                                                        02121
              numberfields                                              02122
              recordsize                                                02123
              overflowaction                                            02124
              underflowaction                                           02125
           %                                                            02126
           LOCAL calprc, value, fld, mode;                              02127
           LOCAL STRING tempsr[50];                                     02128
           REF rd, calprc, fld;                                         02129
           IF initialised THEN RETURN;                                  02130
           fieldnumber _ daughter _  suppressloc _ 0;                   02131
```

```
    IF dynamicrecord THEN                                                02132
       BEGIN                                                             02133
       calldefineproc(&rd);                                              02134
       END                                                               02135
    ELSE                                                                 02136
       BEGIN                                                             02137
       overflowaction _ underflowaction _ proceedaction;                 02138
       replaceaction _ defaultreplace;                                   02139
       numberfields _ recordname.LH;                                     02140
       value _ recordname.RH + recordname.LH-1; %Pointer to last
       field%                                                            02141
       recordsize _ [value].bpadr + 1; %Number of words in record%       02142

       IF numberfields > maxnumfields THEN err($"Record Too
       Large");                                                          02143
       value _ 0;                                                        02144
       mode _ IF macromode AND (numberfields > 1) AND rfnmfg THEN
       $trfname ELSE defaultmode;                                        02145
       &fld _ fieldstart;                                                02146
       DO                                                                02147
          BEGIN                                                          02148
          fieldptr _ [recordname+value];                                 02149
          fieldptr.bpindx _ 0;                                           02150
          typemode _ mode;                                               02151
          &fld _ &fld + fielddescsz;                                     02152
          END UNTIL (value _ value + 1) >= numberfields;                 02153
       END;                                                              02154
    initialised _ TRUE;                                                  02155
    currenttypemode _ symbolicmode; % default printout type %            02156
    RETURN;                                                              02157
    END.                                                                 02158
 (setrd)PROC(rd);                                                        02159
    LOCAL c;                                                             02160
    REF rd;                                                              02161
    c _ -1;                                                              02162
    WHILE (c _ c+1) < rdsize DO rd[c] _ 0;                               02163
    RETURN;                                                              02164
    END.                                                                 02165
%* Special RIB's [dynamic]%                                             02166
   %  General Comments%                                                 02167
    %A record definition procedure recieves one parm, RD.                02168
    It is expected to set up the fields:                                 02169
       Numbrfields                                                       02170
       recordsize,                                                       02171
       overflowaction                                                    02172
       underflowaction                                                   02173
    Plus all of the field descroptors required.                          02174
    It may diddle anything else in the RD which it wishes to%            02175
                                                                         02176
 %Locals%                                                                02177
    (localdef)PROC(rd);                                                  02178
       LOCAL fld, c;                                                     02179
       REF rd, fld;                                                      02180
       fieldnumber _ 0;
       recordsize _ IF framesize <= nparms THEN 0 ELSE  framesize -      02181
       nparms;                                                           02182
       numberfields _MIN(recordsize, maxnumfields);
```

```
        replaceaction _ defaultreplace;                                 02183
        overflowaction _ $bumpframe;                                    02184
        underflowaction _ $preframe;                                    02185
        dynamicrecord _ TRUE;                                           02186
        recordstart _ framep + nparms;                                  02187
        entity _ recordtype;                                            02188
        addressmode _ macro;                                            02189
        &fld _ fieldstart;                                              02190
        c _ 0;                                                          02191
        WHILE (c _ c+1) <= numberfields DO                              02192
           BEGIN                                                        02193
           fieldptr _ wordbyteptr+nparms+c;                             02194
           typemode _ $typlcal;                                         02195
           &fld _ &fld + fielddescsz;                                   02196
           END;                                                         02197
        RETURN;                                                         02198
        END.                                                            02199
     (typlcal)PROC(bp, rd);                                             02200
        LOCAL STRING tempsr[4];                                         02201
        REF rd;                                                         02202
        crlf();                                                         02203
        typech('L);                                                     02204
        *tempsr* _ STRING(fieldnumber);                                 02205
        typeas($tempsr);                                                02206
        typval(.bp, currenttypemode);                                   02207
        RETURN;                                                         02208
        END.                                                            02209
  %Parms%                                                               02210
     (parmdef)PROC(rd);                                                 02211
        LOCAL fld, c;                                                   02212
        REF rd, fld;                                                    02213
        fieldnumber _ 0;                                                02214
        recordsize _ numberfields _ nparms;                             02215
        replaceaction _ defaultreplace;                                 02216
        overflowaction _ $bumpframe;                                    02217
        underflowaction _ $preframe;                                    02218
        dynamicrecord _ TRUE;                                           02219
        recordstart _ framep;                                           02220
        entity _ recordtype;                                            02221
        addressmode _ macro;                                            02222
        &fld _ fieldstart;                                              02223
        c _ 0;                                                          02224
        WHILE (c _ c+1) <= numberfields DO                              02225
           BEGIN                                                        02226
           fieldptr _ wordbyteptr + c;                                  02227
           typemode _ $typparm;                                         02228
           &fld _ &fld + fielddescsz;                                   02229
           END;                                                         02230
        RETURN;                                                         02231
        END.                                                            02232
     (typparm)PROC(bp, rd);                                             02233
        LOCAL STRING tempsr[4];                                         02234
        REF rd;                                                         02235
        crlf();                                                         02236
        typech('P);                                                     02237
        *tempsr* _ STRING(fieldnumber);                                 02238
```

```
        typeas($tempsr);                                                    02239
        typval(.bp, currenttypemode);                                       02240
        RETURN;                                                             02241
        END.                                                                02242
    %Recp [Record Pointer%                                                  02243
      (recpdef)PROC(rd);                                                    02244
        LOCAL fld;                                                          02245
        REF rd, fld;                                                        02246
        recordstart _ $recp;                                                02247
         numberfields _ 2;                                                  02248
        recordsize _ 1;                                                     02249
        overflowaction _ underflowaction _ 0;                               02250
        addressmode _ macro;                                                02251
        replaceaction _ $recprepl;                                          02252
        &fld _ fieldstart;                                                  02253
        fieldptr _ recprecordaddr;                                          02254
        typemode _ $recpt1;                                                 02255
        &fld _ &fld + fielddescsz;                                          02256
        fieldptr _ recprecsize;                                             02257
        typemode _ $recpt2;                                                 02258
        RETURN;                                                             02259
        END.                                                                02260
      (recprepl)PROC(rd);                                                   02261
        LOCAL symtype, symval;                                              02262
        REF rd;                                                             02263
        IF NOT getsym( :symtype, symval)                                    02264
           OR NOT programsymbol                                             02265
            OR NOT ckbptr([symval.RH])                                      02266
           OR symval.LH = 0 THEN                                            02267
              err($"Illegal Record Designator");                            02268
        chkcacr();                                                          02269
        recp _ symval;                                                      02270
        RETURN;                                                             02271
        END.                                                                02272
      (recpt1)PROC(bp, rd);                                                 02273
        LOCAL STRING tempsr[40];                                            02274
        crlf();                                                             02275
        typeas($"First Field in Record is ");                               02276
        ddgtsymbol($tempsr, .bp);                                           02277
        typeas($tempsr);                                                    02278
        RETURN;                                                             02279
        END.                                                                02280
      (recpt2)PROC(bp, rd);                                                 02281
        typech(',);                                                         02282
        typval(.bp, numericmode);                                           02283
        typeas($" Fields in Record");                                       02284
        RETURN;                                                             02285
        END.                                                                02286
    %Stack Frame%                                                           02287
      (bumpframe)PROC(rd);                                                  02288
        REF rd;                                                             02289
        IF framep = frametop THEN err($"Top Of Stack reached");             02290
        framep _ framep + framesize + 2;                                    02291
        initframe();                                                        02292
        RETURN;                                                             02293
```

```
    END.                                                        02294
(framedef)PROC(rd);                                             02295
    LOCAL fld;                                                  02296
    REF rd, fld;                                                02297
    %When called, recordstart contains:                         02298
       1 for frametop                                           02299
       2 for framebase                                          02300
       address for current frame                                02301
    %                                                           02302
    CASE recordstart OF                                         02303
       =frametoptype:                                           02304
          BEGIN                                                 02305
          framep _ frametop;                                    02306
          initframe();                                          02307
          END;                                                  02308
       =framebasetype:                                          02309
          BEGIN                                                 02310
          framep _ framebase;                                   02311
          initframe();                                          02312
          END;                                                  02313
       ENDCASE NULL;                                            02314
    recordstart _ framep;                                       02315
    overflowaction _ $bumpframe;                                02316
    underflowaction _ $preframe;                                02317
    numberfields _ 1;                                           02318
    suppressloc _ TRUE;                                         02319
    fieldnumber _ 0;                                            02320
    recordsize _ framesize;                                     02321
    addressmode _ macro;                                        02322
    setfieldptr;                                                02323
    fieldptr _ 0;                                               02324
    typemode _ $typframe;                                       02325
    RETURN;                                                     02326
    END.                                                        02327
(framfielddef)PROC(rd);                                         02328
    LOCAL fld;                                                  02329
    REF rd, fld;                                                02330
    %When called, recordstart contains:                         02331
       1 for mark                                               02332
       2 fo return                                              02333
       3 for SIGNAL value                                       02334
    %                                                           02335
    overflowaction _ $bumpframe;                                02336
    underflowaction _ $preframe;                                02337
    numberfields _ 1;                                           02338
    suppressloc _ TRUE;                                         02339
    fieldnumber _ 0;                                            02340
    recordsize _ framesize;                                     02341
    addressmode _ macro;                                        02342
    setfieldptr;                                                02343
    fieldptr _ CASE recordstart OF                              02344
       =1: marklocfield;                                        02345
       =2: retlocfield;                                         02346
       =3: siglocfield;                                         02347
       ENDCASE fieldptr; %Change addr call%                     02348
    typemode _ defaultmode;                                     02349
```

```
                                                                    02350
    recordstart _ framep;                                           02351
    RETURN;                                                         02352
    END.                                                            02353
  (initframe)PROC;
    %This procedure sets up the relevant globals for the stack
                                                                    02354
    frame stuff.
       It gets called at initialisation, and ehenever the frame
                                                                    02355
       poiter (framep) gets chnged%                                 02356
    LOCAL t, t1;                                                    02357
    %Find out how many parms%                                       02358
       t _ [returnloc-1];                                           02359
       IF t.opcod10 = opcal10 THEN nparms _ 0                       02360
       ELSE IF t.opcod10 = opcal11 THEN nparms _ 1                  02361
       ELSE IF t.opcod10 = opcallm THEN                             02362
          nparms _ t.accum10 - 2                                    02363
       ELSE nparms _ [returnloc-2].addr10-2;                        02364
    %Get the framesize%                                             02365
       t _ stacktop;
       WHILE [t].marklocfield # framep DO t _ [t].marklocfield;
                                                                    02366
                                                                    02367
       framesize _ t-framep-2;                                      02368
    RETURN;                                                         02369
    END.                                                            02370
  (preframe)PROC(rd);                                               02371
    REF rd;
    IF framep = framebase THEN err($"Bottom of Stack reached");
                                                                    02372
                                                                    02373
    framep _ framemark;                                             02374
    initframe();                                                    02375
    RETURN;                                                         02376
    END.                                                            02377
  (typframe)PROC(bp, rd);                                           02378
    LOCAL STRING tempsr[50];                                        02379
    REF rd;                                                         02380
    %First get current procedure name%                              02381
        ddgtsymbol($tempsr, [returnloc-1].addr10);                  02382
       crlf();                                                      02383
       typeas($"Procedure ");                                       02384
       typeas($tempsr);                                             02385
       crlf();                                                      02386
       ddgtsymbol($tempsr, returnloc-1);                            02387
       typeas($"Called From ");                                     02388
       typeas($tempsr);
       *tempsr* _ STRING(nparms), " Parameters", CR, LF,
       STRING(framesize-nparms), " Locals";                         02389
                                                                    02390
       crlf();                                                      02391
       typeas($tempsr);                                             02392
    RETURN;                                                         02393
    END.                                                            02394
%Strings%                                                           02395
  (replstring)PROC(rd);                                             02744
    LOCAL string;                                                   02397
    REF rd, fld, string;                                            02398
    %First Read the string%                                         02399
       &string _ getstring(2000, $dspblk);                          02400
       txtlit(&string);
```

```
      %Now replace it%                                                  02401
         *[recordstart]* _ *string*;                                    02402
         freestring(&string, $dspblk);                                  02745
      RETURN;                                                           02403
      END.                                                              02404
   (strngrec)PROC(rd);                                                  02405
      LOCAL fld;                                                        02406
      REF rd, fld;                                                      02407
      %Check validity%                                                  02408
         IF [recordstart] = 0                                           02409
         OR [recordstart].M > 3000                                      02410
         OR [recordstart].L > [recordstart].M THEN err($"Illegal
         Format String");                                               02411
      numberfields _ 3;                                                 02412
      recordsize _ ([recordstart].M + 9) / 5;                           02413
      overflowaction _ proceedaction;                                   02414
      underflowaction _ $strngunderflow;                                02415
      addressmode _ macro;                                              02416
      replaceaction _ $replstring;                                      02417
      &fld _ fieldstart;                                                02418
      fieldptr _ stringmax;                                             02419
      typemode _ $typstmx;                                              02420
      &fld _ &fld + fielddescsz;                                        02421
      fieldptr _ stringlength;                                          02422
      typemode _ $typstsz;                                              02423
      &fld _ &fld + fielddescsz;                                        02424
      fieldptr.xptr _ 77B; %Make it an X-pointer%                       02425
      xptrsize _ [recordstart].L;                                       02426
      xptrbytesz _ 7; %Standard A-string alpha%                         02427
      fieldptr.xptadd _ 5; %First character we wish to print%
                                                                        02428
      typemode _ textmode;                                              02429
      RETURN;                                                           02430
      END.                                                              02431
   (strngunderflow)PROC(rd);                                            02432
      LOCAL t, count;                                                   02433
      REF rd;                                                           02434
      t _ recordstart;                                                  02435
      count _ 0;                                                        02436
      WHILE (count _ count + 1) <= maxwordsinstring DO                  02437
         BEGIN                                                          02438
         BUMP DOWN t;                                                   02439
         IF [t] # 0 THEN                                                02440
            IF ((([t].M + 9)/5) = count) AND ([t].L <= [t].M) THEN
                                                                        02441
               BEGIN                                                    02442
               recordstart _ t;                                         02443
               RETURN;                                                  02444
               END;                                                     02445
         END;                                                           02446
      err($"No Preceeding String");                                     02447
      END.                                                              02448
   (typstmx)PROC(bp, rd);                                               02449
      LOCAL STRING tempsr[19];                                          02450
      REF rd;                                                           02451
       *tempsr* _ "    <", STRING(.bp), ':;                             02452
```

```
            typeas($tempsr);                                                02453
            RETURN;                                                         02454
            END.                                                            02455
         (typstsz)PROC(bp, rd);                                             02456
            LOCAL STRING tempsr[19];                                        02457
            REF rd;                                                         02458
             *tempsr* _ STRING(.bp), '>, CR, LF;                            02459
            typeas($tempsr);                                                02460
            RETURN;                                                         02461
            END.                                                            02462
      %Words%                                                               02463
         (wordrec)PROC(rd);                                                 02464
            LOCAL fld;                                                      02465
            REF rd, fld;                                                    02466
            numberfields _ 1;                                               02467
            recordsize _ 1;                                                 02468
            overflowaction _ underflowaction _ proceedaction;               02469
            addressmode _ macro; %Just to make sure%                        02470
            replaceaction _ $replfield;                                     02471
            dynamicrecord _ TRUE; %For next, previous%                      02472
            setfieldptr;                                                    02473
            fieldptr _ wordbyteptr;                                         02474
            typemode _ defaultmode;                                         02475
            RETURN;                                                         02476
            END.                                                            02477
%* Internal Hash Table%                                                    02478
   % Calls %                                                               02479
      (ddtsym) PROC( string );                                              02480
         LOCAL v1,v2,v3;                                                    02481
         LOCAL STRING tempsr[40];                                           02482
         REF string;                                                        02483
         *tempsr* _ *string*;                                               02484
         astruc($tempsr);                                                   02485
         v1 _ hashlookup( $tempsr : v2,v3);                                 02486
         RETURN( v1,v2,v3 );                                                02487
         END.                                                               02488
   %Utilities%                                                              02489
      (ddthash)PROC(string);                                                02490
         LOCAL STRING tempsr[5];                                            02491
         REF string;                                                        02492
         tempsr[1] _ 0;                                                     02493
         IF string.L >= 5 THEN                                              02494
            tempsr[1] _ string[1]                                           02495
         ELSE                                                               02496
            WHILE tempsr.L < string.L DO                                    02497
               *tempsr* _ *tempsr*, *string*[tempsr.L + 1];                 02498
         RETURN((((tempsr[1] .A 35M) / 2) +string.L) MOD hashbase);         02499
                                                                            02500
         END.                                                               02501
      (hashdump)PROC;                                                       02502
         %Type out the contents of te hash table%                           02503
         LOCAL nent, stkusd, maxdepth, hash, t, t1, t2;                     02504
         LOCAL STRING tempsr[100];                                          02505
         hash _ nent _ stkusd _ maxdepth _ 0;                               02506
         LOOP                                                               02507
            BEGIN
```

```
        IF hashstack[hash] # 0 THEN                                     02508
           BEGIN                                                        02509
           BUMP stkusd;                                                 02510
           *tempsr* _ CR, LF, STRING(hash);                             02511
           %typeas($tempsr);%                                           02512
           t _ hashstack[hash];                                         02513
           t2 _ 0;                                                      02514
           DO                                                           02515
              BEGIN                                                     02516
              BUMP nent, t2;                                            02517
              t1 _ t + [t].LH;                                          02518
              *tempsr* _ CR, LF, "     ", *[t.RH+1]*, "( ",
              *[octnum([t1])]*, ", ", *[octnum([t1+1])]*, ");           02519
              %typeas($tempsr);%                                        02520
              END                                                       02521
           UNTIL (t _ [t].RH) = 0;                                      02522
           maxdepth _ MAX(maxdepth, t2);                                02523
           END;                                                         02524
        IF (hash _ hash + 1)>= hashbase THEN EXIT;                      02525
        END;                                                            02526
     %Now type out end stuff - commented out - CFD                      02527
     *xlit* _ CR, LF, CR, LF, "Number Entries = ", *[octnum(nent)]*,
                                                                        02528
        CR, LF, "Maximum Depth = ", *[octnum(maxdepth)]*,               02529
        CR, LF, "Stack Use = ", *[octnum(stkusd)]*, "/",
        *[octnum(hashbase)]*,                                           02530
        CR, LF, "Hash Table Storage = ",
        *[octnum(hashptr-$hashtable)]*, "/", *[octnum(hashtblsz)]*;
                                                                        02531
                                                                        02532
     typeas($xlit); --->>> %                                            02533
     RETURN;                                                            02534
     END.                                                               02535
  (hashenter)PROC(string, value, record, rcdlngth);                     02536
     %enter string if not already entered%                              02537
     LOCAL  hash, index;                                                02538
     REF string;                                                        02539
     hash _ ddthash(&string);                                           02540
     IF hashstack[hash] # 0 THEN                                        02541
        IF hashlookup(&string) THEN RETURN(FALSE); %Duplicate%
     [hashptr].RH _ hashstack[hash]; %Link to next entry in stack%
                                                                        02542
                                                                        02543
     hashstack[hash] _ hashptr;                                         02544
     [hashptr+1].M _ (hashtblsz - 1 - (hashptr-$hashtable))*5;          02545
     *[hashptr+1]* _ *string*;                                          02546
     hashptr _ hashptr + ([hashptr].LH _(string.L + 14)/5);             02547
     [hashptr] _ value;                                                 02548
     BUMP hashptr;                                                      02549
     IF rcdlngth = 1 THEN [hashptr] _ record                            02550
     ELSE mvbfbf(record, hashptr, rcdlngth);                            02551
     hashptr _ hashptr + rcdlngth;                                      02552
     [hashptr] _ 0;                                                     02553
     RETURN;                                                            02554
     END.                                                               02555
  (hashlookup)PROC(string);                                             02556
     LOCAL hash, t, t1;                                                 02557
     REF string;
```

```
    t _ $hashstack +ddthash(&string);                                   02559
    WHILE (t1 _ [t].RH) # 0 DO                                          02560
      BEGIN                                                             02561
      IF *string* = *[t1+1]* THEN                                       02562
        BEGIN                                                           02563
        t _ t1 + [t1].LH;                                               02564
        RETURN(TRUE, [t], t+1);                                         02565
        END;                                                            02566
      t _ t1;                                                           02567
      END;                                                              02568
    RETURN(FALSE);                                                      02569
    END.                                                                02570
(hashtbinit)PROC;                                                       02571
    LOCAL t;                                                            02572
    IF hashptr # 0 THEN RETURN; %Already Initialised%                   02573
    hashptr _ $hashtable;                                               02574
    hashtable _ 0;                                                      02575
    t _ -1;                                                             02576
    WHILE (t_t+1) < hashbase DO hashstack[t] _ 0;                       02577
    RETURN;                                                             02578
    END.                                                                02579
(initht)PROC;                                                           02580
    %INitialise DDT internal Hash Table%                                02581
    LOCAL rdx, rd[ribsize];                                             02582
    IF hashptr # 0 THEN RETURN; %Already DOne%                          02583
    hashtbinit();                                                       02584
    %Do the internal words%                                             02585
      %Do the Registers%                                                02586
        hashenter($"R1", internalsym, 0, 1);                            02587
        hashenter($"R2", internalsym, 1, 1);                            02588
        hashenter($"R3", internalsym, 2, 1);                            02589
        hashenter($"R4", internalsym, 3, 1);                            02590
        hashenter($"R5", internalsym, 4, 1);                            02591
        hashenter($"R6", internalsym, 5, 1);                            02592
        hashenter($"R7", internalsym, 6, 1);                            02593
        hashenter($"PP", internalsym, 7B, 1);                           02594
        hashenter($"S", internalsym, 10B, 1);                           02595
        hashenter($"M", internalsym, 11B, 1);                           02596
        hashenter($"RP", internalsym, 12B, 1);                          02597
        hashenter($"A1", internalsym, 13B, 1);                          02598
        hashenter($"A2", internalsym, 14B, 1);                          02599
        hashenter($"A3", internalsym, 15B, 1);                          02600
        hashenter($"A4", internalsym, 16B, 1);                          02601
      %Now do rest of internal symbols%
        hashenter($"LV", internalsym, $lv + indirectmode-$reg1,
                                                                        02602
        1);
        hashenter($"EC", internalsym, $escchar-$reg1, 1);               02603
        hashenter($"SF", internalsym, $symflg-$reg1, 1);                02604
        hashenter($"RNAMES", internalsym, $rfnmfg-$reg1, 1);
                                                                        02605
        hashenter($"LC", internalsym, $lc + indirectmode-$reg1,
                                                                        02606
        1);                                                             02607
      %Now do te internal records%                                      02608
        rdx _ 0;                                                        02609
        %RECP%                                                          02610
          setrd($rd);
```

```
        recordname _ $recpdef;                                          02611
        dynamicrecord _ TRUE;                                           02612
        entity _ recordtype;                                            02613
        hashenter($"RECP", seqsymbol, $rdx, rdsize+1);                  02614
        hashenter($"R", seqsymbol, $rdx, rdsize+1);                     02615
                                                                        02616
      %Stack Frame Poiter%                                              02617
        setrd($rd);                                                     02618
        recordstart _ $framep;                                          02619
        recordname _ $framedef;                                         02620
        mother _ daughter _ 0;                                          02621
        dynamicrecord _ TRUE;                                           02622
        entity _ recordtype;
        hashenter($"FRAME", seqsymbol + stacksymv, $rdx,
                                                                        02623
        rdsize+1);
        hashenter($"F", seqsymbol + stacksymv, $rdx, rdsize+1);         02624
                                                                        02625
      %PARMATERS%                                                       02626
        setrd($rd);                                                     02627
        recordstart _ $framep;                                          02628
        recordname _ $parmdef;                                          02629
        dynamicrecord _ TRUE;                                           02630
        entity _ recordtype;                                            02631
        hashenter($"PARMS", seqsymbol, $rdx, rdsize+1);                 02632
        hashenter($"P", seqsymbol, $rdx, rdsize+1);                     02633
      %LOCALS%                                                          02634
        setrd($rd);                                                     02635
        recordstart _ $framep;                                          02636
        recordname _ $localdef;                                         02637
        dynamicrecord _ TRUE;                                           02638
        entity _ recordtype;                                            02639
        hashenter($"LOCALS", seqsymbol, $rdx, rdsize+1);                02640
        hashenter($"L", seqsymbol, $rdx, rdsize+1);                     02641
      %Stack top%                                                       02642
        setrd($rd);                                                     02643
        recordstart _ frametoptype;                                     02644
        recordname _ $framedef;                                         02645
        mother _ daughter _ 0;                                          02646
        dynamicrecord _ TRUE;                                           02647
        entity _ recordtype;
        hashenter($"TOP", seqsymbol + stacksymv, $rdx, rdsize+1);       02648
                                                                        02649
      %Frame Base%                                                      02650
        setrd($rd);                                                     02651
        recordstart _ framebasetype;                                    02652
        recordname _ $framedef;                                         02653
        mother _ daughter _ 0;                                          02654
        dynamicrecord _ TRUE;                                           02655
        entity _ recordtype;
        hashenter($"BASE", seqsymbol + stacksymv, $rdx,
                                                                        02656
        rdsize+1);                                                      02657
      %Mark%                                                            02658
        setrd($rd);                                                     02659
        recordstart _ 1;                                                02660
        recordname _ $framfielddef;                                     02661
        mother _ daughter _ 0;                                          02662
        dynamicrecord _ TRUE;
```

```
                entity _ recordtype;                                    02663
                hashenter($"MARK", seqsymbol, $rdx, rdsize+1);          02664
                                                                        02665
             %Returnloc%                                                02666
                setrd($rd);                                             02667
                recordstart _ 2;                                        02668
                recordname _ $framfielddef;                             02669
                mother _ daughter _ 0;                                  02670
                dynamicrecord _ TRUE;                                   02671
                entity _ recordtype;                                    02672
                hashenter($"RET", seqsymbol, $rdx, rdsize+1);           02673
                                                                        02674
             %Signal Loc%                                               02675
                setrd($rd);                                             02676
                recordstart _ 3;                                        02677
                recordname _ $framfielddef;                             02678
                mother _ daughter _ 0;                                  02679
                dynamicrecord _ TRUE;                                   02680
                entity _ recordtype;                                    02681
                hashenter($"SIG", seqsymbol, $rdx, rdsize+1);           02682
          hashdump();                                                   02683
          RETURN;                                                       02684
          END.                                                          02685
%* Utilities%                                                          02686
   (ckbptr)PROC(bp);                                                    02687
      %Checks the validity of a byte ponter...returns False if no good%
                                                                        02688
      IF bp.bpsize > 36                                                 02689
      OR bp.bpbitpos > 36 THEN RETURN(FALSE);                           02690
      RETURN(TRUE);                                                     02691
      END.                                                              02692
   (copyrd)PROC(rd1, rd2);                                              02693
      mvdfbf(rd1, rd2, ribsize);                                        02694
      RETURN;                                                           02743
      END.
FINISH
```

NDOT 2

```
< NLS, NDDT2.NLS;10, >, 9-MAR-77 16:43 JDH ;;;;( NLS, NDDT2.NLS;10, ),
22-MAR-74 05:34 KEV ;
FILE nddt2  %  L10 <rel-NLS>nddt2 %% (L10,) (rel-nls,nddt2.rel,) %     02
   UND-OK                                                              04
   %Defines%                                                           05
      DEFINE strbyteptr =440700000001B#;                              0824
      DEFINE leftbyteptrrj = 4307B8#;                                 0864
      DEFINE bptblentsz = 4#;                                         0867
      DEFINE bptblsz =40#;                                            0868
      DEFINE textmode = 3#;                                           0869
      DEFINE nbkpts=10#;                                              0870
      DEFINE ddtlimit = 3000B#;                                        08
      DEFINE numericmode = 1#;                                        0866
      DEFINE nextchar = binjsys()#;                                    09
      DEFINE backupchar = bkjfnjsys()#;                               010
      DEFINE areg4 =db[15]#;                                          011
      DEFINE lv = db[16]#;                                            0865
      DEFINE reg1 =db#;                                               012
      DEFINE optabl = 24B#;                                           013
      DEFINE loadh = [loadlimit].LH#;                                 015
      DEFINE global = 1#;                                             016
   % Declarations %                                                   017
      DECLARE regnames =("R0  ", "R1  ", "R2  ", "R3  ", "R4  ", "R5  ",
      "R6  ", " P  ", "WA  ", " S  ", " M  ", "RP  ", "A1  ", "A2  ",
      "A3  ", "A4  ");                                                018
      DECLARE symloc = 116B, smanipumask = 77774B7, smanipulator
      =27044B7;                                                       019
      DECLARE EXTERNAL calddt; % location of entry point of NDDT %   0871
      DECLARE ddtsptr=770001B; % ptr to ddt's alt i -1 %              0602
      DECLARE loadlimit = 120B;                                       0298
      REGISTER r1=1, r2=2, r3=3, r4=4;                                0435
      REF ddtsptr;                                                    0603
   % Record definitions %                                            0755
      (inst10)RECORD %PDP10 instruction parts%                        0756
         addr10[18],                                                  0757
         index10[4],                                                  0758
         indir10[1],                                                  0759
         accum10[4],                                                  0760
         opcod10[9];                                                  0761
      (instformat) RECORD pdummy[21], longopcode[15];                 0762
   % Miscellaneous routines used by other NLS procedures %            0763
      (nddtarm)PROC; % called from NLS %                              0764
         LOCAL proc;                                                  0902
         LOCAL STRING nddtname[30];                                   0900
         % Roll in NDDT "user" program. %                            0872
            % Increase buffer size to 6000 free wds. %                0876
              IF NOT gpawdsleft(6000, FALSE) THEN                     0877
                 BEGIN                                                0878
                 err($"Cannot set program buffer for NDDT system!");
                                                                      0879
                 END;                                                 0881
            % Load the user program with NDDT code. %                 0884
              *nddtname* _ "netsys, nddt";                            0885
              ON SIGNAL ELSE                                          0886
                 BEGIN                                                0887
                 ON SIGNAL ELSE;                                      0899
```

```
            err($"Cannot load NDDT program");                          0888
            END;                                                       0890
         getuprog($nddtname);                                          0891
         ON SIGNAL ELSE;                                               0892
       % Get address of "entry point" calldt; put it in acalddt. %     0893
         IF NOT ddtlookup($"CALLDDT", FALSE, % get procedure in
         user program. %: proc) THEN                                   0894
            BEGIN                                                      0895
            err($"Cannot find NDDT entry procedure");                  0896
            END;                                                       0898
         acalddt _ proc;                                               0901
   sethint();                                                          0765
   RETURN;                                                             0766
   END.                                                                0767
(nddtdisarm)PROC; % called from NLS %                                  0768
   unsethint();                                                        0769
   % reset acalddt %                                                   0874
      acalddt _ 0;                                                     0875
   RETURN;                                                             0770
   END.                                                                0771
(sethint)PROC;                                                         0825
   %Set up control H as interrupt%                                     0826
   levtbl _ levtab.RH;                                                 0827
   chntab[5] _ $ddtint .V 1B6;                                         0828
   r1 _ 4B5;                                                           0829
   r2 _ $chntab;                                                       0830
   !HRLI 2,levtab;                                                     0831
   !JSYS sir;                                                          0832
   r1 _ 10000005B;                                                     0833
   !JSYS ati;                                                          0834
   r1 _ 4B5;                                                           0835
   r2 _ 1B10;                                                          0836
   !JSYS aic;                                                          0837
   RETURN;                                                             0838
   %DDT Interrupt handling stuff%                                      0839
      (ddtint):                                                        0840
      !SOS @levtbl; %Point to actual instruction interrupted%          0841
      %Check for stack manipulation instruction%                       0842
         !HRR 0,@levtbl;                                               0843
         !MOVE 0,@0;                                                   0844
         !MOVEM 0,ddtinst; %Save off inst in case we want to
         execute it later%                                             0845
         !AND 0,smanipumask;                                           0846
         !CAME 0,smanipulator;                                         0847
         !JRST ddtnt1;                                                 0848
         %By here, interrupted at stack manipulation
         instruction...execute it%                                     0849
            !XCT ddtinst;                                              0850
            !AOS @levtbl;                                              0851
      (ddtnt1): %Call ddt %                                            0852
         !HRL 0,acalddt;                                               0853
         !HRR 0,@levtbl;                                               0854
         !HLRM 0,@levtbl;                                              0855
         !JSYS debrk;                                                  0856
   END.                                                                0857
```

```
(unsethint)PROC;                                                    0858
   %remove control H as interrupt%                                  0859
   r1 _ 10B;                                                        0860
   !JSYS dti;                                                       0861
   RETURN;                                                          0862
   END.                                                             0863
(octnum)PROC(value);                                                0772
   LOCAL bp;                                                        0773
   %Uses ddtlit%                                                    0774
   bp _ r1 _ strbyteptr + $ddtlit;                                  0775
   r2 _ value;                                                      0776
   r3 _ 8;                                                          0777
   IF NOT SKIP !JSYS nout THEN err($"Error in Nout");               0778
   ddtlit.L _ slngth(bp, r1);                                       0779
   RETURN($ddtlit);                                                 0780
   END.                                                             0781
(tblsearch)PROC(tbl, tblsize, entsize, value);                      0782
   %Generalised table searcher...return TRUE, address of entrym
   and entry number if ok, FALSE otherwise%                         0783
   LOCAL c;                                                         0784
   REF tbl;                                                         0785
   c _ -entsize;                                                    0786
   WHILE (c _ c+entsize) < tblsize DO                               0787
      IF tbl[c] = value THEN RETURN(TRUE, $tbl[c], c/entsize);
                                                                    0788
   RETURN(FALSE);                                                   0789
   END.                                                             0790
(findbp)PROC(value);                                                0791
   %IF value < nbkpts then assume it is a break point number,
   otherwise an address of an instruction%                          0792
   LOCAL bpa, bpn;                                                  0793
   IF value < nbkpts THEN                                           0794
      BEGIN                                                         0795
      bpa _ $bptbl + value*bptblentsz;                              0796
      IF [bpa] = 0 THEN RETURN(0);                                  0797
      RETURN(bpa, value);                                           0798
      END;                                                          0799
   IF NOT tblsearch($bptbl, bptblsz, bptblentsz, value: bpa, bpn)
   THEN RETURN(0);                                                  0800
   RETURN(bpa, bpn);                                                0801
   END.                                                             0802
(typval)PROC(value, mode);                                          0803
   LOCAL bp;                                                        0804
   LOCAL STRING tempsr[75];                                         0805
   REF db;                                                          0903
   lv _ value;                                                      0806
   IF mode = numericmode THEN                                       0807
      IF value.LH # 0 THEN                                          0808
         *tempsr* _ *[octnum(value.LH)]*, ",,",
         *[octnum(value.RH)]*                                       0809
      ELSE                                                          0810
         *tempsr* _ *[octnum(value)]*                               0811
   ELSE IF mode = textmode THEN                                     0812
      BEGIN                                                         0813
      bp _ leftbyteptrrj + $value;                                  0814
      WHILE bp.bpadr = $value DO IF ^bp # 0 THEN typech(.bp);       0815
```

```
        RETURN;                                                    0816
        END                                                        0817
      ELSE                                                         0818
        ddgtsymbol($tempsr, value);                                0819
      typeas($"    ");                                           0820
      typeas($tempsr);                                           0821
      RETURN;                                                    0822
      END.                                                       0823
%* DDT Symbol Table Utility Routines%
                                                                  020
  (ddgtsymbol) % generates a symbolic representation for a value %
                                                                  021
      PROCEDURE(                                                 0579
      % FORMAL PARAMETERS %                                      0582
        astr, % ptr to target character string %                 0580
        value); % fullword value to be converted to symbolic form %
                                                                  0581
      % DETAILED DESCRIPTION OF PROCEDURE %                      0604
        % ALGORITHM %                                            0606
          % if the value word is a pdp10 instruction, then
          instructionmode is set to TRUE and the corresponding
          PDP10 assembly instruction is dis-assembled.  If not,
          then the word is decoded into halfword format if the left
          half is non zero, else it decodes the right half of the
          word %                                                 0620
        % NORMAL RETURNS %                                       0607
          % symbol string is built in the parameter string passed %
                                                                  0616
        % ABNORMAL RETURNS %                                     0608
          % NONE %                                              0617
      % GLOBALS USED %                                          0610
      LOCAL                                                       022
        instructionmode; % set by getins to TRUE if word is a PDP10
        instruction, FALSE otherwise. %                          0618
      REF                                                         023
        astr;                                                    0619
      %-------------%                                           0583
      % set the result string to NULL %                         0584
        *astr* _ NULL;                                            024
      % process left half of value word, checking for inst. %   0585
        IF value.LH # 0 THEN % may be instruction %               025
                                                                  026
          BEGIN                                                   027
          IF NOT (instructionmode _ getins(&astr,value) ) THEN    028
            BEGIN                                                 029
            IF NOT ddsymget(&astr, value.LH) THEN                 030
              *astr* _ *astr*, *[octnum(value.LH)]*;              031
            *astr* _ *astr*, ",,";                                032
            END;                                                  033
          END;                                                    034
      % now do the right half word %                             035
        IF NOT ddsymget(&astr, value.RH) THEN                     036
          *astr* _ *astr*, *[octnum(value.RH)]*;
      % edit index field if PDP10 instruction %                  0586
        IF instructionmode THEN                                   037
          IF value.index10 THEN                                   038
            BEGIN                                                 039
```

```
            *astr* _ *astr*, "(";                                   040
            ddsymget(&astr, value.index10);                         041
            *astr* _ *astr*, ")";                                   042
            END;                                                    043
      RETURN;                                                       044
      END.
                                                                    045
   (ddsymget) % find a DDT symbol associated with a value %         0455
      PROCEDURE(                                                    0587
      % FORMAL PARAMETERS %                                         0589
         astr, % ptr to target char string - result appended to
         string %                                                   0588
         value ); % address value for which symbol is sought %     0590
      % DETAILED DESCRIPTION OF PROCEDURE %                        0621
         % FUNCTION %                                              0629
            % this routine searches the ddt symbol table for a match
            with the given value %                                  0456
         % ALGORITHM %                                             0622
            %  Seccessive blocks of the DDT symbol table are checked
            to see if the target value is represented by some symbol
            in the block.  If so, then the address values of the
            symbol table entries are compared with the target value
            and the symbol having a value closest to the target value
            is selected.  If no symbol is close enough to the target
            symbol (determined by the global value "ddtlimit") then
            FALSE is retruned.  If the closest symbol is a local
            symbol, then the file name is edited into the symbol
            string %                                                0623
            % the search loop has been recoded using inline
            instructions and should be faster than the normal L10
            stuff, but it a bit unreadable %                        0457
            % registers are used by this routine as follows:        0458
               r1    current pointer to symbol value. LH contains -
               length                                               0459
                     remaining in the block                         0460
               r2    sought after symbol table value                0461
               r3    -length of the current symbol table block      0462
               r4    current symbol table test value %              0463
         % NORMAL RETURNS %                                         0624
            % TRUE -- a symbolic representation (symbol + offset) was
            generated in the parameter string %                     0628
            % FALSE -- no appropriate symbol was "close enough" to
            the value to be decoded %                               0625
         % ABNORMAL RETURNS %                                       0626
            % NONE %                                                0627
      LOCAL temp,lowptr,difference, closest;                        0464
      LOCAL blockptr;                                               0465
      LOCAL closestblkptr;                                          0466
      LOCAL STRING tempstr[6];                                      0467
      REF astr, symloc;                                             0468
      %----------%                                                  0591
      % move the compare value to r2 for faster access %           0592
         r2 _ value;                                                0469
      % if value represents a register, no search needed %         0593
         IF r2 IN [$reg1, $areg4] THEN r2 _ r2 - $reg1 + 1;         0470
         IF r2 IN [0, 17B] THEN                                     0471
```

```
        BEGIN %Return a reasonable symbol%                                  0472
        tempstr.L _ 2;                                                      0473
        tempstr[1] _ regnames[r2];                                          0474
        *astr* _ *astr*, *tempstr*;                                         0475
        RETURN (TRUE);                                                      0476
        END;                                                                0477
    % initialize for search through symbol table blocks %                   0594
      lowptr _ symloc.RH; % lowest symbol table address %;                  0478
      blockptr _ (lowptr - symloc.LH).A 18M -1; % highest symbol
      table address %                                                       0479
      difference _ ddtlimit;                                                0480
    % loop through each block in the symbol table %                        0481
      WHILE blockptr > lowptr DO                                            0482
        BEGIN                                                               0483
        !HLRZ r3,@blockptr; % length of block to r3 %                       0484
        r1 _ (blockptr + r3 + 4) .A 18M; % addr of first r2 in
        block %                                                             0485
        % check range of block %                                           0486
        IF r2 INI [blockptr].RH, [r1-2]+ddtlimit] THEN                      0487
          BEGIN                                                             0488
          r1.LH _ r3+4; % effctive range of table %                         0489
          GOTO 11; % start searching up the blocks %                        0490
          (testdifference): % come here to check for a possible
          hit %                                                             0491
            !MOVE r4,0(r1); % get value word from symbol table
            %                                                               0492
            IF r2 - r4 IN [0,difference) THEN                               0493
              BEGIN                                                         0494
              closest _ r1;                                                 0495
              closestblkptr _ blockptr;                                     0496
              IF (difference _ r2 - r4) = 0 THEN GOTO
              exactmatch;                                                   0497
              END;                                                          0498
            GOTO continuesearch;                                            0499
          (11): % top of seach loop %                                       0500
            !CAML r2,0(r1); % compare with r2 %                             0501
            !JRST 0,testdifference;                                         0502
          (continuesearch):                                                 0503
            !AOBJN r1,12; % bump ptr by 1 %                                 0504
          (12):                                                             0505
            !AOBJN r1,11; % fall through loop in no hits %                  0506
          END;                                                              0507
        blockptr _ (blockptr + r3) .A 18M;                                  0508
        END;                                                                0509
    % exit, returning FALSE if no appropriate symbol was found %           0595
      IF difference >= ddtlimit THEN RETURN( FALSE );                       0510
    % generate appropriate symbolic represenation %                        0596
      (exactmatch):                                                         0511
      temp _ [closest.RH-1];                                                0512
      IF temp .A 10B10 THEN % symbol is a LOCAL symbol %                    0513
        BEGIN                                                               0514
        r50toa( [closestblkptr-1] .A 32M, $tempstr); % convert
        block name %                                                        0515
        *astr* _ *astr*, "[, *tempstr*, "]";                                0516
        END;                                                                0517
```

```
      % convert symbol name %                                          0518
      r50toa( temp .A 32M, $tempstr);                                  0519
      *astr* _ *astr*, *tempstr*; % append instruction mnemonic %
                                                                       0520
      IF difference > 0 THEN                                           0521
         *astr* _ *astr*, "+, *[octnum(difference)]*;                  0522
   RETURN( TRUE );                                                     0523
   END.
                                                                       0524
   (getins) PROCEDURE( astr, instruction);                             0630
      LOCAL opcode, i , j, k;                                          0631
      REF astr;                                                        0632
      LOCAL STRING temp[6];                                            0633
      opcode _ instruction.opcod10;                                    0634
      IF NOT( opcode IN [1, 700B]) OR                                  0635
        ((i_optab[opcode]) = 0) THEN RETURN( FALSE );                  0636
      IF i = 1 THEN % indeterminate -look some more %                  0637
         LOOP                                                          0638
            BEGIN                                                      0639
            j _ instruction.longopcode;                                0640
            FOR k _ 0 UP UNTIL >= optabl DO                            0641
               IF optab2[k] = j THEN                                   0642
                  BEGIN                                                0643
                  i _ optab3[k];                                       0644
                  EXIT LOOP 2;                                         0645
                  END;                                                 0646
            RETURN (FALSE);                                            0647
            END;                                                       0648
      % i contains opcode mnemonic in sixbit form %                    0649
      *temp* _ NULL;                                                   0650
      FOR j _ 1 UP UNTIL > 6 DO                                        0651
         BEGIN % convert to ascii by shifting out each digit %
                                                                       0652
         k _ i .A 6M; % pick off right hand 6 bit character %          0653
         *temp* _ k + 32, *temp*;                                      0654
         i _ shr(i, 6); % shift in next digit %                        0655
         END;                                                          0656
      % append to astr %                                              0657
      *astr* _ *astr*, *temp*, "  ";                                   0658
      % edit accumulaor field %                                       0659
      IF instruction.accum10 THEN                                      0660
         BEGIN                                                         0661
         *temp* _ NULL;                                                0662
         ddsymget( $temp, instruction.accum10);                        0663
         *astr* _ *astr*, *temp*, ",;                                  0664
         END;                                                          0665
      % indirect flag %                                               0666
      IF instruction.indir10 THEN *astr* _ *astr*, "@;                 0667
      RETURN ( TRUE );                                                 0668
      END.                                                             0669
      (shl)  PROCEDURE( value, shiftcount );                           0670
         % this routine performs the logical left shift of the
         specified number %                                            0671
         r1 _ value;                                                   0672
         r2 _ 242040B6 + (shiftcount .A 18M); %construct a LSH
         instruction in r2 %                                           0673
```

```
        !XCT r2; % execute the shift - result left in r1 %          0674
        RETURN (r1);                                                0675
        END.                                                        0676
     (shr) PROCEDURE( value, shiftcount );                          0677
        % this routine performs the logical right shift of the
        specified mumber %                                          0678
        r1 _ value;                                                 0679
        r2 _ 242040B6 + ((0-shiftcount) .A 18M); %construct a LSH
        instruction in r2 %                                         0680
        !XCT r2; % execute the shift - result left in r1 %          0681
        RETURN (r1);                                                0682
        END.                                                        0683
(r50toa) PROCEDURE( value, astr);                                   0312
   % this procedure converts a RADIX50 symbol representation into
   a NLS string %                                                   0313
   LOCAL j;                                                         0314
   REF astr;                                                        0315
   *astr* _ NULL;                                                   0316
   LOOP % convert from radix 50 to ascii %                          0317
      BEGIN                                                         0318
      DIV value/50B, value, j;                                      0319
      IF NOT j OR astr.L = astr.M THEN EXIT LOOP;                   0320
      j _ IF (j _ j + 47) > 57 THEN j+7 ELSE j;                     0321
      *astr* _ j , *astr*;                                          0322
      END;                                                          0323
   RETURN;                                                          0324
   END.                                                             0325
(ddtenter) PROCEDURE (astr, value, type);                           046
   % this routine enters a symbol and value into the ddt symbol
   table
   the current range of the symbol table is contained in "symloc"
   the highest load address is contained in "loadh". %             047
   LOCAL ptr, i, blockptr;                                          048
   REF symloc ,astr, blockptr;                                      049
   % find end of current block -- look for a word of all zeroes %
                                                                    050
   &blockptr _ symloc.RH + 2;                                       051
   UNTIL blockptr = 0 DO                                            052
      &blockptr _ &blockptr + 2; % try next entry %                 053
   &blockptr _ &blockptr - 1;                                       054
   % LH of [blockptr] ccntaines -1 where 1 is the length of the
   block%                                                           055
   ptr _ (&blockptr + blockptr.LH -1) .A 18M;                       056
   % check for table overflow %                                    057
   IF ptr < loadh THEN                                              058
      err( $"Symbol Table Overflow" ); % out of space %            0434
   % update symloc to reflect new extent of table %               059
   symloc _ symloc - 2000002B;                                      060
   % increase length by -2 in header word %                        061
   blockptr _ blockptr - 2B6;                                       062
   % make new header for block
     first word is zeroes
     second word contains the highest address in the block %       063
   [ptr] _ 0;                                                       064
   BUMP ptr;                                                        065
   [ptr] _  MAX([ptr+2],value);                                     066
```

```
    % add new entry to block %                                          067
    BUMP ptr;                                                           068
    [ptr] _ ddtname( &astr, type);                                      069
    BUMP ptr;                                                           070
    [ptr] _ value;                                                      071
    RETURN( TRUE );                                                     072
    END.                                                                073
(ddtlookup) PROCEDURE(astr, skiplocals);                                074
    % IF the second argument is TRUE then the mark stack is ignored
    and the entire DDT symbol table is searched bottom up %             075
    LOCAL i,j,v,blockptr,address,value;                                 076
    REF astr, symloc;                                                   077
    % convert symbol to a radix 50 value %                             078
    v _ ddtname(&astr, global);                                         079
    % now search down through the symbol table %                       080
                                                                        081
    IF NOT skiplocals THEN                                              082
       FOR i _ ddmrkx-1 DOWN UNTIL < 0 DO                               083
          % search through marked table for local symbols first %
                                                                        084
          IF ddtsrchblk( ddmrk[i], v .X 14B10: value,address)           085
             % search for global symbols in any marked blocks next
             %                                                          086
             OR ddtsrchblk( ddmrk[i], v: value,address)                 087
             THEN RETURN(TRUE, value, address);                         088
                                                                        089
    % now search the whole damn table %                                090
    j _ symloc.RH; % lowest symblol table address %                     091
    blockptr _ (j - symloc.LH) .A 18M - 1; % higheat sym table
    address %                                                           092
    WHILE blockptr > j DO                                               093
       BEGIN                                                            094
       IF ddtsrchblk( blockptr, v : value, address) THEN                095
          RETURN( TRUE, value, address);                                096
       blockptr _ (blockptr + [blockptr].LH) .A 18M;                    097
       END;                                                             098
    RETURN(FALSE);                                                      099
    END.                                                               0100
(mrkglookup) PROCEDURE(astr);                                          0739
    % Looks for the symbol specified by astr as a global in the
    mark stack%                                                        0740
    %used by gpget to get the subsystem dispatch record given the
    subsystem name%                                                    0741
    LOCAL i,j,v,blockptr,address,value;                                0742
    REF astr, symloc;                                                  0743
    % convert symbol to a radix 50 value %                            0744
    v _ ddtname(&astr, 1);                                             0745
    % now search down through the symbol table %                      0746
                                                                       0747
    FOR i _ ddmrkx-1 DOWN UNTIL < 0 DO                                 0748
       % search for global symbols in any marked blocks first %
                                                                       0749
       IF ddtsrchblk( ddmrk[i], v: value,address)                      0750
          THEN RETURN(TRUE, value, address);                           0751
                                                                       0752
    RETURN(FALSE);                                                     0753
```

```
    END.                                                              0754
(ddtmark) PROCEDURE( astr );                                          0699
    % this routine markes the current position in the ddtsymbol
    table and creates a new block with the name given the value
    saved in the mark stack is the block pointer for the marked
    block%                                                            0700
    LOCAL namevalue,ptr1,blockptr;                                    0702
    REF symloc, astr;                                                 0703
    % search through the ddt symbol table looking for the named
    block %                                                           0704
    namevalue _ ddtname( &astr, 0) .A 32M;                            0705
    blockptr _ (symloc.RH - symloc.LH - 1) .A 18M;                    0706
    LOOP BEGIN                                                        0707
        IF [blockptr-1] = namevalue THEN EXIT LOOP % found block %
                                                                      0708
        ELSE BEGIN                                                    0709
            blockptr _ (blockptr + [blockptr].LH ) .A 18M;            0710
            IF blockptr <= symloc.RH THEN                             0711
                BEGIN % not found -- must allocate a new one %        0712
                % allocate 4 words for new header is there is room %
                                                                      0713
                symloc _ symloc - 40000048;                           0714
                ptr1 _ symloc.RH;                                     0715
                IF ptr1 < [loadlimit].LH THEN err($"No More Room For
                Symbol Table");                                       0716
                % initialize the 4 word header %                     0717
                [ptr1] _ 0;                                           0718
                BUMP ptr1;                                            0719
                [ptr1] _ 0;                                           0720
                BUMP ptr1;                                            0721
                [ptr1] _ namevalue; % radix50 name %                  0722
                BUMP ptr1;                                            0723
                [ptr1].LH _ -4;                                       0724
                [ptr1].RH _ [ptr1+2];                                 0725
                EXIT LOOP;                                            0726
                END;                                                  0727
            END;                                                      0728
        END;                                                          0729
    ddmrk[ddmrkx] _ blockptr; % save away current blockptr %          0730
    IF (ddmrkx _ ddmrkx + 1) > ddmrkm THEN                            0731
        BEGIN                                                         0732
        ddmrkx _ ddmrkm;                                              0733
        err($"Symbol Table Mark Stack Overflowed");                   0734
        END;                                                          0735
    RETURN;                                                           0736
    END.                                                              0737
                                                                      0738
(ddtname) PROCEDURE( astr, type);                                     0141
    % this routine returns a radix 50 interpretation of the string
    and incorporates the type information with it to produce a
    ddtsymbol.  Only Upper or Lower case alphabetic and numeric
    characters are allowed. %                                         0142
    LOCAL value, i ,j;                                                0144
    REF astr;                                                         0145
    % make radix 50 value from symbol %                              0146
    value _ 0;                                                        0147
```

```
      FOR i _ 1 UP UNTIL > MIN(astr.L,6) DO                          0148
         BEGIN                                                       0149
         j _ CASE j_*astr*[i] OF                                     0691
            INC 60B, 71B]:   % digits %                              0692
               j - 57B;                                              0693
            INC101B,132B]:   % Upper case alpha %                    0694
               j - 66B;                                              0695
            INC141B,172B]:   % lower case alpha %                    0696
               j - 126B;                                             0697
            ENDCASE err($"Invalid char in identifier");              0698
         value _ 40 * value + j;                                     0152
         END;                                                        0153
      % set appropriate type mask %                                  0154
      value _ value .V (IF type = global THEN 4B10 ELSE 10B10);      0155
      RETURN (value);                                                0156
      END.                                                           0157
   (ddtsrchblk) PROCEDURE( blockptr, symbolhash);                    0436
      % this  routine searches through a specified block of the
      symbol table, looking for a match with a given symbol value.
      Inline coding has been used to speed up the search loop and
      registers are used as follows:                                 0437
         r1  pointer to current location in the symbol table         0438
         r2  test value                                              0439
         r3  temporary value %                                       0440
      !HLRZ r3,@blockptr; % get length of symbol table block to r3 %
                                                                     0441
      r1 _ (blockptr + r3 + 3);                                      0442
      !ADDI r3,4;                                                    0443
      !HRL r1,r3; % set length of r1 %                               0444
      r2 _ symbolhash; % load up with symbol hash value %            0445
      IF [r1-2] # 0 THEN err( $" bad blockptr in ddtsrchblk ");      0446
      (srch1): !CAMN r2,0(r1); % compare %                           0447
         !JRST 0,srchhit; % found it %                               0448
         !AOBJN r1,srch2;                                            0449
      (srch2): !AOBJN r1,srch1; % loop to srch1 if any left to
      compare %                                                      0450
      RETURN (FALSE);                                                0451
      (srchhit): % found the symbol %                                0452
         RETURN( TRUE, [r1+1], r1+1)                                 0453
      END.                                                           0454
   (ddtpop) PROCEDURE;                                               0169
      % this routine pops the current routine from the marked symbol
      table stack %                                                  0170
      % decrement stack index and error check %                      0171
      IF (ddmrkx _ ddmrkx - 1) < 0 THEN                              0172
         BEGIN                                                       0173
         ddmrkx _ 0;                                                 0174
         err($"Symbol Table Pop Stack Error");                       0175
         END;                                                        0176
      % recover symbol table space if at the end of the table %     0177
      ddtchkp( ddmrk[ddmrkx] );                                      0178
      RETURN;                                                        0179
      END.                                                           0180
   (ddtchkp) PROC(blockptr);                                         0181
      LOCAL ddtloc;                                                  0601
      REF symloc;                                                    0182
```

```
        % this procedure checks to see if the symbol table block
        defined by blockptr is at the bottom of the DDT symbol table,
        and if so, it recovers the space by modifying symloc
        appropriately %                                              0183
        IF (blockptr + [blockptr].LH + 1).A 18M = symloc.RH          0184
           THEN BEGIN                                                0185
           symloc.LH _ symloc.LH - [blockptr].LH;                    0186
           symloc.RH _ blockptr + 1;                                 0187
           % reset ddt's alt i -1 pointer also %                     0598
              ddtloc _ ddtsptr;                                      0599
              [ddtloc] _ symloc;                                     0600
           END;                                                      0188
        RETURN;                                                      0189
        END.                                                         0190
FINISH                                                                03
```

NDOT DATA.

```
< NLS, NDDTDATA.NLS;5, >, 7-APR-76 14:57 DSM ;;;;
FILE nddtdata   % L10   <REL-NLS>NDDTDATA   %% (L10,)
(rel-nls,NDDTDATA.rel,) %                                                  02

                                                                           03
REGISTER %here so DDT will use these on printout%                          04
   r1=1, r2=2, r3=3, r4=4, r5=5, p=7, wa=8, s=9, m=10, rp=11,              05
   a1=12, a2=13, a3=14, a4=15;                                             06
% NDDT support %                                                          07
   DECLARE EXTERNAL                                                        08
      ddmrk[21],          % mark stack %                                   09
      ddmrkm=20,          % max stack depth %                              010
      ddmrkx=0, % current stack depth %                                    011
      db,                 % ptr to current data block %                   012
   % Breakpoint support %                                                  013
      bptbl[40],          %breakpoint table nbkpts * bpentsz%             014
      ddtrloc,  % breakpoint address %                                     015
      ddtrp,    % trap address %                                          016
      ddtinst,  % interrupted instruction %                                017
      ddtjunk,  % temporaries for processing breakpoints %                 018
      ddtrg1,                                                              019
      ddtrg2,                                                              020
      ddtrg3,                                                              021
   % NDDT hash symbol table %                                             022
      hsymtb[300],        % symbol table space %                           023
      hstkhed[19],        % hash table %                                   024
      hsymptr,  % pointer to symbol table entry %                          025
   % table of replaced procedures %                                       026
      proctbl[30],                                                         027
   % misc. data %                                                         034
      acalddt=0, % contains address of NDDT entry point or 0 %             028
      levtbl,   % control-h interrupt table %
      displayflag;        % flag for saving and restoring of display areas
      %                                                                    029
                                                                           030
   DECLARE EXTERNAL STRING                                                 031
      ddtlit[150];        % general temp char string %                     032
   DECLARE EXTERNAL STRING                                                 035
      dnstr[50];  %used by dn%                                             033
FINISH of nddtdata
```

NLSSRT

```
< NLS, NLSSRT.NLS;2, >, 2-OCT-74 12:37 HGL ;;;;                    038
;Output assembler file to (nls, nlssrt.mac,)                       02

search stenex                                                      03
title nlssrt                                                       04
                                                                   05
; defined possible entry points                                   034
   nls==0        ;main entry point                                 028
   tnls==2       ;tnls entry point                                 029
   dex==3        ;dex entry point                                  030
   jback==4      ;journal background entry point                   031
   nic==5        ;query entry point                                032
                                                                   037
; make sure entloc is defined                                      035
   ifndef entloc,<entloc==0>                                       036
                                                                   033
start:                                                             06
   hrroi 2,[asciz /<netsys>nls.sav/]                               013
   hrizi 1,100001                                                  014
   gtjfn                                                           015
   haltf                                                           016
   move 2,[XWD geter,3]                                            017
   blt 2,6                                                         018
   hrli 1,400000                                                   019
   jrst 3                                                          020
geter:                                                             021
   get                                                             022
   movei 1,400000                                                  023
   gevec                                                           024
   jrst entloc(2)                                                  025
                                                                   026
   end start                                                       027
```

OPTAB

```
< NLS, OPTAB.NLS;1, >, 9-SEP-76 10:46 SKO ;;;;                          0430
;this file was made  by a copy sequential from nls,optab.mac             02
;<DORNBUSH>SAVE.NLS;6, 30-NOV-72 9:32 CFD ;                              03
   INTERN      OPTAB,OPTAB2,OPTAB3                                       04
OPTAB:  REPEAT  40,<EXP 0>                  ; 0 - 37                     05
   SIXBIT      "CALL  "        ; 40                                      06
   SIXBIT      "INIT  "        ; 41                                      07
   REPEAT      5,<EXP 0>                   ; 42 - 46                    08
   SIXBIT      "CALLI "        ; 47                                      09
   SIXBIT      "OPEN  "        ; 50                                     010
   SIXBIT      "TTCALL"        ; 51                                     011
   REPEAT      3,<EXP 0>                   ; 52 - 54                    012
   SIXBIT      "RENAME"        ; 55                                     013
   SIXBIT      "IN    "        ; 56                                     014
   SIXBIT      "OUT   "        ; 57                                     015
   SIXBIT      "SETSTS"        ; 60                                     016
   SIXBIT      "STATO "        ; 61                                     017
   SIXBIT      "STATUS"        ; 62                                     018
   SIXBIT      "STATZ "        ; 63                                     019
   SIXBIT      "INBUF "        ; 64                                     020
   SIXBIT      "OUTBUF"        ; 65                                     021
   SIXBIT      "INPUT "        ; 66                                     022
   SIXBIT      "OUTPUT"        ; 67                                     023
   SIXBIT      "CLOSE "        ; 70                                     024
   SIXBIT      "RELEAS"        ; 71                                     025
   SIXBIT      "MTAPE "        ; 72                                     026
   SIXBIT      "UGETF "        ; 73                                     027
   SIXBIT      "USETI "        ; 74                                     028
   SIXBIT      "USETO "        ; 75                                     029
   SIXBIT      "LOOKUP"        ; 76                                     030
   SIXBIT      "ENTER "        ; 77                                     031
   REPEAT      30,<EXP 0>                  ; 100 - 127                  032
   SIXBIT      "UFA   "        ; 130                                    033
   SIXBIT      "DFN   "        ; 131                                    034
   SIXBIT      "FSC   "        ; 132                                    035
   SIXBIT      "IBP   "        ; 133                                    036
   SIXBIT      "ILDB  "        ; 134                                    037
   SIXBIT      "LDB   "        ; 135                                    038
   SIXBIT      "IDPB  "        ; 136                                    039
   SIXBIT      "DPB   "        ; 137                                    040
   SIXBIT      "FAD   "        ; 140                                    041
   SIXBIT      "FADL  "        ; 141                                    042
   SIXBIT      "FADM  "        ; 142                                    043
   SIXBIT      "FADB  "        ; 143                                    044
   SIXBIT      "FADR  "        ; 144                                    045
   SIXBIT      "FADRI "        ; 145                                    046
   SIXBIT      "FADRM "        ; 146                                    047
   SIXBIT      "FADRB "        ; 147                                    048
   SIXBIT      "FSB   "        ; 150                                    049
   SIXBIT      "FSBL  "        ; 151                                    050
   SIXBIT      "FSBM  "        ; 152                                    051
   SIXBIT      "FSBB  "        ; 153                                    052
   SIXBIT      "FSBR  "        ; 154                                    053
   SIXBIT      "FSBRI "        ; 155                                    054
   SIXBIT      "FSBRM "        ; 156                                    055
   SIXBIT      "FSBRB "        ; 157
```

```
SIXBIT      "FMP   "      ; 160                                    056
SIXBIT      "FMPL  "      ; 161                                    057
SIXBIT      "FMPM  "      ; 162                                    058
SIXBIT      "FMPB  "      ; 163                                    059
SIXBIT      "FMPR  "      ; 164                                    060
SIXBIT      "FMPRI "      ; 165                                    061
SIXBIT      "FMPRM "      ; 166                                    062
SIXBIT      "FMPRB "      ; 167                                    063
SIXBIT      "FDV   "      ; 170                                    064
SIXBIT      "FDVL  "      ; 171                                    065
SIXBIT      "FDVM  "      ; 172                                    066
SIXBIT      "FDVB  "      ; 173                                    067
SIXBIT      "FDVR  "      ; 174                                    068
SIXBIT      "FDVRI "      ; 175                                    069
SIXBIT      "FDVRM "      ; 176                                    070
SIXBIT      "FDVRB "      ; 177                                    071
SIXBIT      "MOVE  "      ; 200                                    072
SIXBIT      "MOVEI "      ; 201                                    073
SIXBIT      "MOVEM "      ; 202                                    074
SIXBIT      "MOVES "      ; 203                                    075
SIXBIT      "MOVS  "      ; 204                                    076
SIXBIT      "MOVSI "      ; 205                                    077
SIXBIT      "MOVSM "      ; 206                                    078
SIXBIT      "MOVSS "      ; 207                                    079
SIXBIT      "MOVN  "      ; 210                                    080
SIXBIT      "MOVNI "      ; 211                                    081
SIXBIT      "MOVNM "      ; 212                                    082
SIXBIT      "MOVNS "      ; 213                                    083
SIXBIT      "MOVM  "      ; 214                                    084
SIXBIT      "MOVMI "      ; 215                                    085
SIXBIT      "MOVMM "      ; 216                                    086
SIXBIT      "MOVMS "      ; 217                                    087
SIXBIT      "IMUL  "      ; 220                                    088
SIXBIT      "IMULI "      ; 221                                    089
SIXBIT      "IMULM "      ; 222                                    090
SIXBIT      "IMULB "      ; 223                                    091
SIXBIT      "MUL   "      ; 224                                    092
SIXBIT      "MULI  "      ; 225                                    093
SIXBIT      "MULM  "      ; 226                                    094
SIXBIT      "MULB  "      ; 227                                    095
SIXBIT      "IDIV  "      ; 230                                    096
SIXBIT      "IDIVI "      ; 231                                    097
SIXBIT      "IDIVM "      ; 232                                    098
SIXBIT      "IDIVB "      ; 233                                    099
SIXBIT      "DIV   "      ; 234                                   0100
SIXBIT      "DIVI  "      ; 235                                   0101
SIXBIT      "DIVM  "      ; 236                                   0102
SIXBIT      "DIVB  "      ; 237                                   0103
SIXBIT      "ASH   "      ; 240                                   0104
SIXBIT      "ROT   "      ; 241                                   0105
SIXBIT      "LSH   "      ; 242                                   0106
SIXBIT      "JFFO  "      ; 243                                   0107
SIXBIT      "ASHC  "      ; 244                                   0108
SIXBIT      "ROTC  "      ; 245                                   0109
SIXBIT      "LSHC  "      ; 246                                   0110
Z                         ; 247                                   0111
```

```
SIXBIT      "EXCH  "        ; 250                                   0112
SIXBIT      "BLT   "        ; 251                                   0113
SIXBIT      "AOBJP "        ; 252                                   0114
SIXBIT      "AOBJN "        ; 253                                   0115
EXP  1                      ; 254                                   0116
EXP  1                      ; 255                                   0117
SIXBIT      "XCT   "        ; 256                                   0118
Z                           ; 257                                   0119
SIXBIT      "PUSHJ "        ; 260                                   0120
SIXBIT      "PUSH  "        ; 261                                   0121
SIXBIT      "POP   "        ; 262                                   0122
SIXBIT      "POPJ  "        ; 263                                   0123
SIXBIT      "JSR   "        ; 264                                   0124
SIXBIT      "JSP   "        ; 265                                   0125
SIXBIT      "JSA   "        ; 266                                   0126
SIXBIT      "JRA   "        ; 267                                   0127
SIXBIT      "ADD   "        ; 270                                   0128
SIXBIT      "ADDI  "        ; 271                                   0129
SIXBIT      "ADDM  "        ; 272                                   0130
SIXBIT      "ADDB  "        ; 273                                   0131
SIXBIT      "SUB   "        ; 274                                   0132
SIXBIT      "SUBI  "        ; 275                                   0133
SIXBIT      "SUBM  "        ; 276                                   0134
SIXBIT      "SUBB  "        ; 277                                   0135
SIXBIT      "CAI   "        ; 300                                   0136
SIXBIT      "CAIL  "        ; 301                                   0137
SIXBIT      "CAIE  "        ; 302                                   0138
SIXBIT      "CAILE "        ; 303                                   0139
SIXBIT      "CAIA  "        ; 304                                   0140
SIXBIT      "CAIGE "        ; 305                                   0141
SIXBIT      "CAIN  "        ; 306                                   0142
SIXBIT      "CAIG  "        ; 307                                   0143
SIXBIT      "CAM   "        ; 310                                   0144
SIXBIT      "CAML  "        ; 311                                   0145
SIXBIT      "CAME  "        ; 312                                   0146
SIXBIT      "CAMLE "        ; 313                                   0147
SIXBIT      "CAMA  "        ; 314                                   0148
SIXBIT      "CAMGE "        ; 315                                   0149
SIXBIT      "CAMN  "        ; 316                                   0150
SIXBIT      "CAMG  "        ; 317                                   0151
SIXBIT      "JUMP  "        ; 320                                   0152
SIXBIT      "JUMPL "        ; 321                                   0153
SIXBIT      "JUMPE "        ; 322                                   0154
SIXBIT      "JUMPLE"        ; 323                                   0155
SIXBIT      "JUMPA "        ; 324                                   0156
SIXBIT      "JUMPGE"        ; 325                                   0157
SIXBIT      "JUMPN "        ; 326                                   0158
SIXBIT      "JUMPG "        ; 327                                   0159
SIXBIT      "SKIP  "        ; 330                                   0160
SIXBIT      "SKIPL "        ; 331                                   0161
SIXBIT      "SKIPE "        ; 332                                   0162
SIXBIT      "SKIPLE"        ; 333                                   0163
SIXBIT      "SKIPA "        ; 334                                   0164
SIXBIT      "SKIPGE"        ; 335                                   0165
SIXBIT      "SKIPN "        ; 336                                   0166
SIXBIT      "SKIPG "        ; 337                                   0167
```

```
SIXBIT      "AOJ   "    ; 340                                   0168
SIXBIT      "AOJL  "    ; 341                                   0169
SIXBIT      "AOJE  "    ; 342                                   0170
SIXBIT      "AOJLE "    ; 343                                   0171
SIXBIT      "AOJA  "    ; 344                                   0172
SIXBIT      "AOJGE "    ; 345                                   0173
SIXBIT      "AOJN  "    ; 346                                   0174
SIXBIT      "AOJG  "    ; 347                                   0175
SIXBIT      "AOS   "    ; 350                                   0176
SIXBIT      "AOSL  "    ; 351                                   0177
SIXBIT      "AOSE  "    ; 352                                   0178
SIXBIT      "AOSLE "    ; 353                                   0179
SIXBIT      "AOSA  "    ; 354                                   0180
SIXBIT      "AOSGE "    ; 355                                   0181
SIXBIT      "AOSN  "    ; 356                                   0182
SIXBIT      "AOSG  "    ; 357                                   0183
SIXBIT      "SOJ   "    ; 360                                   0184
SIXBIT      "SOJL  "    ; 361                                   0185
SIXBIT      "SOJE  "    ; 362                                   0186
SIXBIT      "SOJLE "    ; 363                                   0187
SIXBIT      "SOJA  "    ; 364                                   0188
SIXBIT      "SOJGE "    ; 365                                   0189
SIXBIT      "SOJN  "    ; 366                                   0190
SIXBIT      "SOJG  "    ; 367                                   0191
SIXBIT      "SOS   "    ; 370                                   0192
SIXBIT      "SOSL  "    ; 371                                   0193
SIXBIT      "SOSE  "    ; 372                                   0194
SIXBIT      "SOSLE "    ; 373                                   0195
SIXBIT      "SOSA  "    ; 374                                   0196
SIXBIT      "SOSGE "    ; 375                                   0197
SIXBIT      "SOSN  "    ; 376                                   0198
SIXBIT      "SOSG  "    ; 377                                   0199
SIXBIT      "SETZ  "    ; 400                                   0200
SIXBIT      "SETZI "    ; 401                                   0201
SIXBIT      "SETZM "    ; 402                                   0202
SIXBIT      "SETZB "    ; 403                                   0203
SIXBIT      "AND   "    ; 404                                   0204
SIXBIT      "ANDI  "    ; 405                                   0205
SIXBIT      "ANDM  "    ; 406                                   0206
SIXBIT      "ANDB  "    ; 407                                   0207
SIXBIT      "ANDCA "    ; 410                                   0208
SIXBIT      "ANDCAI"    ; 411                                   0209
SIXBIT      "ANDCAM"    ; 412                                   0210
SIXBIT      "ANDCAB"    ; 413                                   0211
SIXBIT      "SETM  "    ; 414                                   0212
SIXBIT      "SETMI "    ; 415                                   0213
SIXBIT      "SETMM "    ; 416                                   0214
SIXBIT      "SETMB "    ; 417                                   0215
SIXBIT      "ANDCM "    ; 420                                   0216
SIXBIT      "ANDCMI"    ; 421                                   0217
SIXBIT      "ANDCMM"    ; 422                                   0218
SIXBIT      "ANDCMB"    ; 423                                   0219
SIXBIT      "SETA  "    ; 424                                   0220
SIXBIT      "SETAI "    ; 425                                   0221
SIXBIT      "SETAM "    ; 426                                   0222
SIXBIT      "SETAB "    ; 427                                   0223
```

```
SIXBIT      "XOR   "      ; 430                                    0224
SIXBIT      "XORI  "      ; 431                                    0225
SIXBIT      "XORM  "      ; 432                                    0226
SIXBIT      "XORB  "      ; 433                                    0227
SIXBIT      "IOR   "      ; 434                                    0228
SIXBIT      "IORI  "      ; 435                                    0229
SIXBIT      "IORM  "      ; 436                                    0230
SIXBIT      "IORB  "      ; 437                                    0231
SIXBIT      "ANDCB "      ; 440                                    0232
SIXBIT      "ANDCBI"      ; 441                                    0233
SIXBIT      "ANDCBM"      ; 442                                    0234
SIXBIT      "ANDCBB"      ; 443                                    0235
SIXBIT      "EQV   "      ; 444                                    0236
SIXBIT      "EQVI  "      ; 445                                    0237
SIXBIT      "EQVM  "      ; 446                                    0238
SIXBIT      "EQVB  "      ; 447                                    0239
SIXBIT      "SETCA "      ; 450                                    0240
SIXBIT      "SETCAI"      ; 451                                    0241
SIXBIT      "SETCAM"      ; 452                                    0242
SIXBIT      "SETCAB"      ; 453                                    0243
SIXBIT      "ORCA  "      ; 454                                    0244
SIXBIT      "ORCAI "      ; 455                                    0245
SIXBIT      "ORCAM "      ; 456                                    0246
SIXBIT      "ORCAB "      ; 457                                    0247
SIXBIT      "SETCM "      ; 460                                    0248
SIXBIT      "SETCMI"      ; 461                                    0249
SIXBIT      "SETCMM"      ; 462                                    0250
SIXBIT      "SETCMB"      ; 463                                    0251
SIXBIT      "ORCM  "      ; 464                                    0252
SIXBIT      "ORCMI "      ; 465                                    0253
SIXBIT      "ORCMM "      ; 466                                    0254
SIXBIT      "ORCMB "      ; 467                                    0255
SIXBIT      "ORCB  "      ; 470                                    0256
SIXBIT      "ORCBI "      ; 471                                    0257
SIXBIT      "ORCBM "      ; 472                                    0258
SIXBIT      "ORCBB "      ; 473                                    0259
SIXBIT      "SETO  "      ; 474                                    0260
SIXBIT      "SETOI "      ; 475                                    0261
SIXBIT      "SETOM "      ; 476                                    0262
SIXBIT      "SETOB "      ; 477                                    0263
SIXBIT      "HLL   "      ; 500                                    0264
SIXBIT      "HLLI  "      ; 501                                    0265
SIXBIT      "HLLM  "      ; 502                                    0266
SIXBIT      "HLLS  "      ; 503                                    0267
SIXBIT      "HRL   "      ; 504                                    0268
SIXBIT      "HRLI  "      ; 505                                    0269
SIXBIT      "HRLM  "      ; 506                                    0270
SIXBIT      "HRLS  "      ; 507                                    0271
SIXBIT      "HLLZ  "      ; 510                                    0272
SIXBIT      "HLLZI "      ; 511                                    0273
SIXBIT      "HLLZM "      ; 512                                    0274
SIXBIT      "HLLZS "      ; 513                                    0275
SIXBIT      "HRLZ  "      ; 514                                    0276
SIXBIT      "HRLZI "      ; 515                                    0277
SIXBIT      "HRLZM "      ; 516                                    0278
SIXBIT      "HRLZS "      ; 517                                    0279
```

```
SIXBIT      "HLLO  "      ; 520                                    0280
SIXBIT      "HLLOI "      ; 521                                    0281
SIXBIT      "HLLOM "      ; 522                                    0282
SIXBIT      "HLLOS "      ; 523                                    0283
SIXBIT      "HRLO  "      ; 524                                    0284
SIXBIT      "HRLOI "      ; 525                                    0285
SIXBIT      "HRLOM "      ; 526                                    0286
SIXBIT      "HRLOS "      ; 527                                    0287
SIXBIT      "HLLE  "      ; 530                                    0288
SIXBIT      "HLLEI "      ; 531                                    0289
SIXBIT      "HLLEM "      ; 532                                    0290
SIXBIT      "HLLES "      ; 533                                    0291
SIXBIT      "HRLE  "      ; 534                                    0292
SIXBIT      "HRLEI "      ; 535                                    0293
SIXBIT      "HRLEM "      ; 536                                    0294
SIXBIT      "HRLES "      ; 537                                    0295
SIXBIT      "HRR   "      ; 540                                    0296
SIXBIT      "HRRI  "      ; 541                                    0297
SIXBIT      "HRRM  "      ; 542                                    0298
SIXBIT      "HRRS  "      ; 543                                    0299
SIXBIT      "HLR   "      ; 544                                    0300
SIXBIT      "HLRI  "      ; 545                                    0301
SIXBIT      "HLRM  "      ; 546                                    0302
SIXBIT      "HLRS  "      ; 547                                    0303
SIXBIT      "HRRZ  "      ; 550                                    0304
SIXBIT      "HRRZI "      ; 551                                    0305
SIXBIT      "HRRZM "      ; 552                                    0306
SIXBIT      "HRRZS "      ; 553                                    0307
SIXBIT      "HLRZ  "      ; 554                                    0308
SIXBIT      "HLRZI "      ; 555                                    0309
SIXBIT      "HLRZM "      ; 556                                    0310
SIXBIT      "HLRZS "      ; 557                                    0311
SIXBIT      "HRRO  "      ; 560                                    0312
SIXBIT      "HRROI "      ; 561                                    0313
SIXBIT      "HRROM "      ; 562                                    0314
SIXBIT      "HRROS "      ; 563                                    0315
SIXBIT      "HLRO  "      ; 564                                    0316
SIXBIT      "HLROI "      ; 565                                    0317
SIXBIT      "HLROM "      ; 566                                    0318
SIXBIT      "HLROS "      ; 567                                    0319
SIXBIT      "HRRE  "      ; 570                                    0320
SIXBIT      "HRREI "      ; 571                                    0321
SIXBIT      "HRREM "      ; 572                                    0322
SIXBIT      "HRRES "      ; 573                                    0323
SIXBIT      "HLRE  "      ; 574                                    0324
SIXBIT      "HLREI "      ; 575                                    0325
SIXBIT      "HLREM "      ; 576                                    0326
SIXBIT      "HLRES "      ; 577                                    0327
SIXBIT      "TRN   "      ; 600                                    0328
SIXBIT      "TLN   "      ; 601                                    0329
SIXBIT      "TRNE  "      ; 602                                    0330
SIXBIT      "TLNE  "      ; 603                                    0331
SIXBIT      "TRNA  "      ; 604                                    0332
SIXBIT      "TLNA  "      ; 605                                    0333
SIXBIT      "TRNN  "      ; 606                                    0334
SIXBIT      "TLNN  "      ; 607                                    0335
```

```
SIXBIT      "TDN   "    ; 610                                    0336
SIXBIT      "TSN   "    ; 611                                    0337
SIXBIT      "TDNE  "    ; 612                                    0338
SIXBIT      "TSNE  "    ; 613                                    0339
SIXBIT      "TDNA  "    ; 614                                    0340
SIXBIT      "TSNA  "    ; 615                                    0341
SIXBIT      "TDNN  "    ; 616                                    0342
SIXBIT      "TSNN  "    ; 617                                    0343
SIXBIT      "TRZ   "    ; 620                                    0344
SIXBIT      "TLZ   "    ; 621                                    0345
SIXBIT      "TRZE  "    ; 622                                    0346
SIXBIT      "TLZE  "    ; 623                                    0347
SIXBIT      "TRZA  "    ; 624                                    0348
SIXBIT      "TLZA  "    ; 625                                    0349
SIXBIT      "TRZN  "    ; 626                                    0350
SIXBIT      "TLZN  "    ; 627                                    0351
SIXBIT      "TDZ   "    ; 630                                    0352
SIXBIT      "TSZ   "    ; 631                                    0353
SIXBIT      "TDZE  "    ; 632                                    0354
SIXBIT      "TSZE  "    ; 633                                    0355
SIXBIT      "TDZA  "    ; 634                                    0356
SIXBIT      "TSZA  "    ; 635                                    0357
SIXBIT      "TDZN  "    ; 636                                    0358
SIXBIT      "TSZN  "    ; 637                                    0359
SIXBIT      "TRC   "    ; 640                                    0360
SIXBIT      "TLC   "    ; 641                                    0361
SIXBIT      "TRCE  "    ; 642                                    0362
SIXBIT      "TLCE  "    ; 643                                    0363
SIXBIT      "TRCA  "    ; 644                                    0364
SIXBIT      "TLCA  "    ; 645                                    0365
SIXBIT      "TRCN  "    ; 646                                    0366
SIXBIT      "TLCN  "    ; 647                                    0367
SIXBIT      "TDC   "    ; 650                                    0368
SIXBIT      "TSC   "    ; 651                                    0369
SIXBIT      "TDCE  "    ; 652                                    0370
SIXBIT      "TSCE  "    ; 653                                    0371
SIXBIT      "TDCA  "    ; 654                                    0372
SIXBIT      "TSCA  "    ; 655                                    0373
SIXBIT      "TDCN  "    ; 656                                    0374
SIXBIT      "TSCN  "    ; 657                                    0375
SIXBIT      "TRO   "    ; 660                                    0376
SIXBIT      "TLO   "    ; 661                                    0377
SIXBIT      "TROE  "    ; 662                                    0378
SIXBIT      "TLOE  "    ; 663                                    0379
SIXBIT      "TROA  "    ; 664                                    0380
SIXBIT      "TLOA  "    ; 665                                    0381
SIXBIT      "TRON  "    ; 666                                    0382
SIXBIT      "TLON  "    ; 667                                    0383
SIXBIT      "TDO   "    ; 670                                    0384
SIXBIT      "TSO   "    ; 671                                    0385
SIXBIT      "TDOE  "    ; 672                                    0386
SIXBIT      "TSOE  "    ; 673                                    0387
SIXBIT      "TDOA  "    ; 674                                    0388
SIXBIT      "TSOA  "    ; 675                                    0389
SIXBIT      "TDON  "    ; 676                                    0390
SIXBIT      "TSON  "    ; 677                                    0391
```

```
   EXP  1                  ; 700                                          0392
OPTAB2: EXP     25400                                                     0393
   EXP  25410                                                            0394
   EXP  25420                                                            0395
   EXP  25450                                                            0396
   EXP  25500                                                            0397
   EXP  25504                                                            0398
   EXP  25510                                                            0399
   EXP  25520                                                            0400
   EXP  25530                                                            0401
   EXP  25540                                                            0402
   EXP  70000                                                            0403
   EXP  70004                                                            0404
   EXP  70010                                                            0405
   EXP  70014                                                            0406
   EXP  70020                                                            0407
   EXP  70024                                                            0408
   EXP  70030                                                            0409
   EXP  70034                                                            0410
OPTAB3: SIXBIT  "JRST  "        ; 25400                                  0411
   SIXBIT       "JRSTF "        ; 25410                                  0412
   SIXBIT       "HALT  "        ; 25420                                  0413
   SIXBIT       "JEN   "        ; 25450                                  0414
   SIXBIT       "JFCL  "        ; 25500                                  0415
   SIXBIT       "JFOV  "        ; 25504                                  0416
   SIXBIT       "JCRY1 "        ; 25510                                  0417
   SIXBIT       "JCRY0 "        ; 25520                                  0418
   SIXBIT       "JCRY  "        ; 25530                                  0419
   SIXBIT       "JOV   "        ; 25540                                  0420
   SIXBIT       "BLKI  "        ; 70000                                  0421
   SIXBIT       "DATAI "        ; 70004                                  0422
   SIXBIT       "BLKO  "        ; 70010                                  0423
   SIXBIT       "DATAO "        ; 70014                                  0424
   SIXBIT       "CONO  "        ; 70020                                  0425
   SIXBIT       "CONI  "        ; 70024                                  0426
   SIXBIT       "CONSZ "        ; 70030                                  0427
   SIXBIT       "CONSO "        ; 70034                                  0428
   END                                                                   0429
```

PARSEK

```
< NLS, PARSER.NLS;35, >, 14-MAR-77 14:22 KJM ;;;;
FILE parser % L10 <rel-nls>parser.rel %% (L10,) (rel-nls,parser.rel,) %
                                                                    02
% DOCUMENTATION %                                                  03
   % control environment for the interpreter %                     04
      % the interpreter is controlled by two processing stacks: the
      "path stack" and the "eval stack".  The path stack records the
      path through the grammar and contains values retruned from
      processing funtions.  The eval stack holds pointers to path stack
      records corresponding to the dynamic evaluation of the parser.
      The interpreter is a stack machine, and values for function actual
      parameters and pointers to the return value records are contained
      in the eval stack %                                           05
      % the functions recognized by the interpreter cause pointers to
      path stack records to be popped and pushed onto the eval stack %
                                                                    06
   % the use of interpreter variables %                           07
      % interpreter variables are data cells which contain pointers to
      path stack records.  A check is made whenever a variable is
      referenced that the pointer is still valid %                  08
% DECLARATIONS %                                                   09
   % REGISTERS %                                                  010
      REGISTER r1 = 1, r2 = 2, r3 = 3, r4 = 4, p = 7, m = 10, s = 9; 011
   % REF VARIABLES %                                             0185
      REF fbstr, inpt;                                          0186
      REF tda, sysmsg, hlpcmdstk;                               0187
% EXECUTION CONTROL ROUTINES %                                   0188
   (initsubsystems) PROC; %* defines the "normal" subsystems available
   from NLS %                                                   05869
      % RETURNS %                                               05870
         % none %                                               05871
      LOCAL % VARIBLES %                                        05872
         numwsubsys,    %number or word for each subsystem/program name%
                                                                05873
         subptr,        %ptr to string containing subsystem/program
         name%                                                  05874
         sdptr, %ptr to subsystem dispatch record%              05875
         notlevel1,     % ctrl bits for not level 1 name %      05876
         ctrlbits;      % interpreter contol bits for keyword
         recognition %                                          05877
      REF subptr;                                               05878
      % ----------------- %                                     05879
      % NOTE *** the first subsystem defined here becomes the "base
      level" or default subsystem for NLS %                     05880
      ctrlbits _ 7B; % Level 1 + DNLS + TNLS %                  05881
      notlevel1 _ 3B; % DNLS + TNLS %                           05882
      % define all subsystems %                                05883
         dfnsubsys( $nlseditor, ctrlbits ,$allsubs);          %
         nlseditor %                                            05884
         %dfnsubsys( $subcalculator, ctrlbits ,$allsubs);%    %
         subcalculator %                                        05885
         %dfnsubsys( $subident, ctrlbits ,$allsubs);%         %
         subident %                                             05887
         %                                                      05888
         dfnsubsys( $subreadmail, ctrlbits ,$allsubs);        %% read
         mail %%                                                05889
```

```
      %                                                                  05890
      dfnsubsys( $subsendmail, ctrlbits ,$allsubs);          % send
      mail %                                                             05891
      dfnsubsys( $subsyntax, notlevel1 ,$allsubs);           % syntax
      generator %                                                        06557
      dfnsubsys( $subprograms, ctrlbits ,$allsubs);          % user
      subprograms %                                                      05892
      dfnsubsys( $subuser, ctrlbits ,$allsubs);              %
      user-options %                                                     05893
      dfnsubsys( $subxxx, ctrlbits ,$allsubs);               %
      systems personnel subsystem %                                      05922
      dfnsubsys( $subsupervisor, notlevel1 ,$allsubs);
      % user-options %                                                   05894
   %define the subsystems specified in user option page%                05895
      %check to see if this user's options are set up%                  05896
      numwsubsys _ ((usys1.M + 4) / 5) + 1;                            05898
      FOR &subptr _ $usys2 UP numwsubsys UNTIL > $usys15 DO            05899
         IF subptr.L THEN                                              05900
            IF sdptr _ getsdptr(&subptr,$allsubs) THEN                 05901
               dfnsubsys(sdptr,gtctrlbits(sdptr, $nlssubs),$nlssubs)
                                                                       05919
            ELSE                                                       05902
               BEGIN                                                   05906
               ON SIGNAL                                               05907
               ELSE                                                    05908
                  BEGIN                                                05910
                  ON SIGNAL ELSE;                                      05921
                  dismes(2,&sysmsg);                                   05909
                  REPEAT LOOP;                                         05911
                  END;                                                 05912
               getuprog( &subptr);                                     05905
               ON SIGNAL ELSE;                                         05920
               END;                                                    05913
   RETURN;                                                             05903
   END.

                                                                       05904
(getsdptr)   %gets a ptr to s subsystem dispatch record%               09753
   PROCEDURE (                                                         09754
   % FORMAL ARGUEMENTS%                                                09755
      stptr, %ptr to a string containing subsystem name%               09756
      subrule));      %ptr to a subsystem rule  nlssubs or allsubs%
                                                                       09757
   % RETURNS %                                                         09758
      % ptr to the appropiate subsystem dispatch record  or FALSE %
                                                                       09759
   LOCAL   %Variables%                                                 09760
      sdptr, %ptr to the appropiate subsystem dispatch record%         09761
      frameptr;       %ptr to current alternative of the rule subrule%
                                                                       09762
   LOCAL TEXT POINTER txa, txb, txc;                                   09763
   LOCAL STRING ssname[25];                                            09764
   REF %Variables%                                                     09765
      stptr, %ptr to a string containing subsystem name%               09766
      subrule,                                                         09767
      frameptr;                                                        09768
   %extract subsystem name from link%                                  09769
```

```
      IF NOT FIND SF(*stptr*) > [',] ^txa THEN FIND SF(*stptr*) ^txa;
                                                                    09770
      IF NOT FIND SE(*stptr*) < [ '. / '; ] ^txb THEN FIND
      SE(*stptr*) ^txb;                                             09771
      *ssname* _ + txa txb;                                         09772
   % set up frameptr and check to see we have really been passed a
   valid rule%                                                      09773
      &frameptr _ &subrule;                                         09774
      IF frameptr.opcode NOT= $xecute THEN RETURN(FALSE);           09775
      &frameptr _ frameptr.addr;                                    09776
      WHILE &frameptr DO                                            09777
         IF *[frameptr.addr]* = *ssname* THEN                       09778
            BEGIN                                                   09779
            % the ptr to the subsystem dispatch record is the address
            field of the sucessorof this frame%                     09780
            &frameptr _ frameptr.nsuccessor;                        09781
            sdptr _ frameptr.addr;                                  09782
            RETURN(sdptr)                                           09783
            END                                                     09784
         ELSE &frameptr _ frameptr.alternative;                     09785
      RETURN(FALSE);                                                09786
   END.                                                             09787
(gotohelp) PROCEDURE;                                               06436
   % save the accumulators %                                        06437
   svac1 _ r1; r1 _ $svacs; !BLT r1, svacse;                        06438
   s _ s + 40000040B;                                               06439
   enthelp( FALSE);                                                 06440
   !HRLZI r1, svacs;                                                06441
   !BLT r1, 17B;                                                    06442
   r1 _ svac1;                                                      06443
   !JSYS debrk;                                                     06444
   END.
                                                                    06445
(enthelp) PROCEDURE ( mode % If TRUE we are in help already % ); %
set up to $enter the subhelp system after a ^Q %                    06558
   LOCAL % VARIABLES %                                              06559
      subsysptrloc,  % dummy params for $call to xgoto %            06560
      dispatchptrloc, % ditto %                                     06561
      exflagptrloc,  % ditto %                                      06562
      entryptr,      % ptr to current subsystem stack entry %       06563
      nextptr,       % ptr to bottom of subsystem stack %           06564
      curptr;        % ptr to path stack entry %                    06565
   REF entryptr, nextptr, curptr;                                   06566
   % hlpcmdstk REFd at start of file %                              06567
   % this routine sets up the global string pointers hlpcmdstk to
   point to the place into the place in the data base to which the
   user is taken.  It then fakes an EXECUTE HELP so that the user
   will end up in the subhelp system.  Note that we do not go
   directly there, but end up there when the next input character was
   read %                                                           06568
   %-----------%                                                    06569
   % set up ptr to subsystem name %                                06570
      &entryptr _ $sbstack + sbstkx - $sbentsize;                   06571
   IF NOT mode THEN                                                 06572
      BEGIN                                                         06573
      % Get storage for name stack %                               06574
```

```
      IF NOT &hlpcmdstk THEN &hlpcmdstk _ getarray(hpcmdmax + 1,
      $dspblk);                                                    06575
   %  set ptr to current command (if there is one) %              06576
      &curptr _ $pathstk;                                          06577
      hlpcmdstk _ 1;                                               06578
      hlpcmdstk[1] _ entryptr.sbnptr;                              06579
      IF pathx > 0 % not at command reset state % THEN             06580
         BEGIN                                                     06581
         &nextptr _ $pathstk + pathx;                              06582
         % get the rest of the path %                              06583
         DO                                                        06584
            BEGIN                                                  06585
            CASE curptr.pfuncton OF                                06586
               = $keywrec:                                         06587
                  BEGIN                                            06588
                  BUMP hlpcmdstk;                                  06589
                  hlpcmdstk[hlpcmdstk] _ [curptr.curnodeptr].addr;
                                                                   06590
                  END;                                             06591
               = $xselect: NULL;                                   06592
                                                                   06593
                  % BEGIN                                          06594
                  hlpcmdstk _ 2;                                   06595
                  hlpcmdstk[1] _ $"operands";                      06596
                  hlpcmdstk[2] _                                   06597
                     CASE [curptr.curnodeptr].opcode OF            06598
                        = $ssel: $"source";                        06599
                        = $dsel: $"destination";                   06600
                        = $lsel: $"content";                       06601
                        ENDCASE $"source";                         06602
                  END; %                                           06603
                                                                   06604
               = $xviewspecs:                                      06605
                  BEGIN                                            06606
                  hlpcmdstk _ 2;                                   06607
                  hlpcmdstk[2] _ $"viewspecs";                     06609
                  END;                                             06610
               = $xlevadj:                                         06611
                  BEGIN                                            06612
                  hlpcmdstk _ 2;                                   06613
                  hlpcmdstk[2] _ $"level-adjust";                  06614
                  END;                                             06616
               = $xconfirm:                                        06617
                  BEGIN                                            06618
                  hlpcmdstk _ 2;                                   06619
                  hlpcmdstk[2] _ $"ok";                            06620
                  END;                                             06621
               ENDCASE;                                            06622
            &curptr _ &curptr + $totalrecsize;                     06623
            END                                                    06624
         UNTIL (&curptr >= &nextptr) OR (hlpcmdstk = hpcmdmax);
                                                                   06625
         END;                                                      06626
   END;                                                            06627
% set up to execute the help rule (with the current subsystem name
visible) %                                                         06628
```

```
      % Fake dispatch record: current subsystem name, apprpriate help
      rule. %                                                    06629
         subsysptrloc _ helploc.dptname _ entryptr.sbnptr; %
         subsystem name %                                        06630
         helploc.dptrun _  IF nlmode = fulldisplay THEN $qdhelp  06631
            ELSE $qthelp; % rule to be executed %                06632
         helploc.dptvalid _ $dptvldationcode;                    06633
         helploc.dptinit _ helploc.dptfinish _  helploc.dptnotused _
         helploc.dptrentry _ 0;                                  06634
      dispatchptrloc _ $helploc;                                 06635
      exflagptrloc _ 1; % Flag for execution rather than goto. %
                                                                 06636
      xgoto(0, parsing, $subsysptrloc, $dispatchptrloc,
      $exflagptrloc);                                            06637
   % stuff a cmdelete character into the input buffer %          06638
      !sti(18M, CD);                                             06639
   % resume execution %                                         06640
      RETURN;                                                    06641
   END.
                                                                 06642
(dfnsubsys) PROCEDURE( % defines a subsystem by adding/replacing a
subsystem to/from the passed subsystem dispatch stack %          05550
   % FORMAL ARGUMENTS %                                         05551
      dptptr,         % ptr to subsystem dispatch record (this is the
      symbolic name which appears on the SUBSYSTEM statement of the
      CML definition of the subsystem %                          05552
      controlbits,    % TNLS, DNLS, level 1 options %           05553
      subrule);       %ptr to subsystem rule (either nlssubs or
      allsubs)%                                                  05554
   % RETURNS %                                                  05555
      % none %                                                  05556
   % ABNORMAL RETURNS %                                         05557
      % SIGNALS interperr if stack overflows %                  05558
   LOCAL % VARIABLES %                                         05559
      frameptr,       % ptr to current frame in the allsubs %   05560
      instptr,        % ptr to CML instruction record %         05561
      pdptx,          %ptr to size of subrule %                05562
      i;      % index counter %                                 05563
   REF % VARIABLES %                                           05564
      dptptr, frameptr, instptr, pdptx, subrule;                 05565
   % -------------------- %                                     05566
   % check to make sure that we've been given a valid pointer % 05567
      IF dptptr.dptvalid # $dptvldationcode THEN                 05568
         err( $"Invalid Subsystem Identifier");                  05569
   %check to be sure we are passed a valid subrule and set up pdptx%
                                                                 05570
      CASE &subrule OF                                           05571
         = $nlssubs: &pdptx _ $sdptx;                            05572
         = $allsubs: &pdptx _ $sdptxa;                           05573
         ENDCASE err($"Invalid subsystem rule passed to dfnsubsys");
                                                                 05574
   % initialize the subsystem dispatch stack if it has not already
   been done %                                                   05575
      IF pdptx < 4 THEN                                          05576
         BEGIN                                                   05577
         &instptr _ &subrule; % build $xecute instruction %      05578
```

```
      FOR &frameptr _ &instptr UP UNTIL > &instptr+3 DO                05579
         % initialize the first 4 words to zeroes %                    05580
           frameptr _ 0;                                               05581
      % build an execute instruction in the first position %           05582
         instptr.opcode _ $xecute;                                     05583
      % build a STORE insturction in the next position %               05584
         &instptr _ &instptr + 2;                                      05585
         instptr.opcode _ $store;                                      05586
         instptr.addr _ $grammar;                                      05587
      % set pdptx to point to next loc in dispatch stack %             05588
         pdptx _ 4;                                                    05589
      END;                                                             05590
% search through the dispatch stack and set frameptr to point to
the frame to be replaced.  If none is found, look for a null frame
(from a previous deleted subsystem) %                                  05591
   FOR &frameptr _ &subrule+4 UP $framesize UNTIL >=
   &subrule+pdptx DO                                                   05592
      CASE frameptr.opcode OF                                          05593
         = $keyop:        % try to replace existing subsystem %
                                                                       05594
           IF *[frameptr.addr]* = *[dptptr.dptname]* THEN              05595
              BEGIN                                                    05596
              &instptr _ &frameptr;                                    05597
              instptr.addr _ dptptr.dptname;                           05598
              instptr.ctrl _ controlbits;                              05599
              &instptr _ &instptr + 2;                                 05600
              instptr.addr _ &dptptr;                                  05601
              RETURN;                                                  05602
              END;                                                     05603
      ENDCASE;                                                         05604
% search through the dispatch stack and set frameptr to point to
a null frame (from a previous deleted subsystem) %                     05605
   FOR &frameptr _ &subrule+4 UP $framesize UNTIL >=
   &subrule+pdptx DO                                                   05606
      CASE frameptr.opcode OF                                          05607
         = 0:     % try to replace null frame %                       05608
           GOTO buildit;                                               05609
      ENDCASE;                                                         05610
% allocate a new frame at the end of the stack %                       05611
   &frameptr _ &subrule+pdptx;                                         05612
   IF pdptx >= $sdptsize THEN                                          05613
      SIGNAL(interperr, $"Subsystem dispatch stack overflowed");
                                                                       05614
   pdptx _ pdptx + $framesize;                                         05615
(buildit):                                                             05616
% initialize the frame to 0 %                                         05617
   FOR &instptr _ &frameptr UP UNTIL >= &frameptr+$framesize DO
                                                                       05618
      instptr _ 0;                                                     05619
% build a keyword recognition instruction - set the successor to
point to the next instruction (to be subsequently built) and set
the alternative path to zero %                                         05620
   &instptr _ &frameptr;                                               05621
   instptr.nsuccessor _ &frameptr+2;                                   05622
   instptr.opcode _ $keyop;                                            05623
   instptr.addr _ dptptr.dptname;                                      05624
```

```
      instptr.ctrl _ controlbits;                                    05625
   % build an $enter value instruction - the alternative path is null
   as this is the last inst. in the stack.  The successor points to
   the STORE instruction in the top of the stack %                   05626
      &instptr _ &frameptr+2;                                        05627
      instptr.nsuccessor _ &subrule + 2;                             05628
      instptr.opcode _ $enter;                                       05629
      instptr.addr _ &dptptr;                                        05630
   % link the new entry into the existing stack.  If this is the
   first entry on the stack, then the EXECUTE instruction at the top
   of the stack must point to here, otherwise this frame is linked as
   an alternative to the previous frame %                            05631
      IF &frameptr = &subrule + $framesize                           05632
         THEN % first entry %                                        05633
            BEGIN                                                    05634
            frameptr.alternative _ subrule.addr;                     05635
            subrule.addr _ &frameptr                                 05636
            END                                                      05637
         ELSE                                                        05638
            BEGIN % subsequent entry %                               05639
            &instptr _ &frameptr - $framesize;                       05640
            frameptr.alternative _ instptr.alternative;              05641
            instptr.alternative _ &frameptr;                         05642
            END;                                                     05643
   RETURN;                                                           05644
   END.
                                                                     05645
(delsubsys) PROCEDURE( % deletes a subsystem by nulling its entry in
the subsystem rule subrule %                                         04980
   % FORMAL ARGUMENTS %                                              04981
      dptptr,          % ptr to subsystem dispatch record (this is the
      symbolic name which appears on the SUBSYSTEM statement of the
      CML definition of the subsystem %                              04982
      subrule);        %ptr to a subsystem rule (nlssubs or allsubs)%
                                                                     04983
   % RETURNS %                                                       04984
      % TRUE if successful, FALSE subsystem not found in the nlssubs
      stack %                                                        04985
   % ABNORMAL RETURNS %                                             04986
      % calls err if actual argument is not valid %                 04987
   LOCAL % VARIABLES %                                               04988
      frameptr,        % ptr to current frame in the nlssubs %       04989
      instptr,         % ptr to CML instruction record %            04990
      savalt,          %used to save alternative of deleted entry%
                                                                     04991
      i;     % index counter %                                      04992
   REF % VARIABLES %                                                 04993
      dptptr, frameptr, subrule, instptr;                            04994
   % -------------------- %                                         04995
   % check to make sure that we've been given a valid pointer %     04996
      IF dptptr.dptvalid # $dptvldationcode THEN                     04997
         err( $"Invalid Subsystem Identifier");                      04998
   % search through the dispatch stack and set frameptr to point to
   the frame to be deleted. %                                        04999
      &frameptr _ &subrule;                                          05000
      IF frameptr.opcode NOT= $xecute THEN RETURN( FALSE );          05001
```

```
      &frameptr _ frameptr.addr;                                        05002
      IF *[frameptr.addr]* = *[dptptr.dptname]* THEN                    05003
         BEGIN                                                          05004
         subrule.addr _ frameptr.alternative;                           05005
         % zap the four word entry to zeoes %                           05006
            FOR &instptr _ &frameptr UP UNTIL >= &frameptr+$framesize
            DO                                                          05007
               instptr _ 0;                                             05008
         RETURN( TRUE );                                                05009
         END;                                                           05010
      WHILE &frameptr DO                                                05011
         IF frameptr.alternative AND [frameptr.alternative].opcode =
         $keyop THEN                                                    05012
            IF *[[frameptr.alternative].addr]* = *[dptptr.dptname]*
            THEN                                                        05013
               BEGIN                                                    05014
               savalt _ [frameptr.alternative].alternative;             05015
               % zap the four word entry to zeoes %                     05016
                  FOR &instptr _ frameptr.alternative UP UNTIL >=
                  frameptr.alternative+$framesize DO                    05017
                     instptr _ 0;                                       05018
               frameptr.alternative _savalt;                            05019
               RETURN( TRUE );                                          05020
               END                                                      05021
            ELSE &frameptr _ frameptr.alternative                       05022
         ELSE RETURN( FALSE );                                          05023
   RETURN (FALSE);                                                      05024
   END.
                                                                        05025
(supervisor) PROC; % main control routine %                             08351
   % supervisor is the principal control routine for NLS.  It
   maintains a stack of subsystem executions and passes control to
   the command interpreter %                                            08352
   % FORMAL ARGUMENTS %                                                08353
      % none %                                                         08354
   % RETURN VALUES %                                                   08355
      % none -- this routine maintains control until we quit NLS
      altogether, at which time it simply returns to its caller %
                                                                        08356
   LOCAL % VARIABLES %                                                  08357
      loadavg,       %system load average%                             08358
      sdispptr,      % ptr to subsystem dispatch record %              08359
      count, % iteration count for subsystem %                          08360
      entry, % ptr to current sbstack entry %                           08361
      ptr,           % starting point for cmdinterpretter %            08362
      instptr,       % ptr to dummy interpretter instructions %        08363
      insts[4];      % dummy grammar to connect supervisor to
      subsystem %                                                       08364
   LOCAL STRING                                                         08365
      loadstr[20],          %formatted load average string%           08366
      locstr[150],          %used for file names%                      08367
      sysnamestr[50];       % subhelp string for subsystems %          08368
   REF % VARIABLES %                                                    08369
      notetime,                                                         08370
      entry,                                                            08371
      instptr,                                                          08372
```

```
   sdispptr;                                                    08373
LOCAL STRING errstr[70];                                        08374
%-------------%                                                 08375
% initialize the NLS subsystems if they have not already been
initialized .  If we get an error here, just abort by returning %
                                                                08376
   ON SIGNAL                                                    08377
      ELSE RETURN;                                              08378
   IF sdptx <= 0 THEN initsubsystems();                         08379
% initialize the system message pointer to null %              08380
   &sysmsg _ 0;                                                 08381
% initialize some interpreter control stuff %                  08382
   sbstkx _ 0;                                                  08383
% construct an initial sbstack entry:  Use the first entry in the
NLSSUB stack of subsystems as the default subsystem %           08384
   &instptr _ $nlssub+6; % ptr to first ENTER instruction %     08385
   % check to make sure we've really got an $enter instruction %
                                                                08386
      IF instptr.opcode # $enter THEN RETURN; % abort %         08387
   &sdispptr _ instptr.addr;                                    08388
   &entry _ $sbstack;                                           08389
   entry.sbptr _ &sdispptr;                                     08390
   entry.sbcount _ -1;                                          08391
   entry.sbmode _ sbstart;                                      08392
   entry.sbnptr _ sdispptr.dptname;                             08393
   sbstkx _ $sbentsize;                                         08394
% initialize the insts to connect a subsystem grammar with the
supervisor's grammar %                                          08395
   CASE &sdispptr _ getsdptr($usys1,$allsubs) OF                08396
      = 0: %not an nls subsystem must be a user subsystem%      08397
         BEGIN                                                  08398
            ON SIGNAL                                           08399
            ELSE                                                08400
               BEGIN                                            08401
               ON SIGNAL ELSE;                                  08402
               *errstr* _ *usys1* , "  not available: using nls
               default supervisor";                             08403
               dismes(2,$errstr);                               08404
               &sdispptr _ $subsupervisor;                      08405
               EXIT CASE;                                       08406
               END;                                             08407
            getuprog($usys1);                                   08408
            IF &sdispptr _ getsdptr($usys1,$nlssubs) THEN       08409
               delsubsys(&sdispptr,$nlssubs)%gpget has defined
               supervisor as a subsystem , we can't allow user to
               go to his supervisor subsystem%                  08410
            ELSE err($errstr); %the specified user program was not
            a subsystem%                                        08411
         END;                                                   08412
   ENDCASE;                                                     08413
   FOR &instptr _ $insts UP UNTIL = $insts+4 DO instptr _ 0;    08414
   &instptr _ $insts;                                           08415
   instptr.opcode _ $xecute;                                    08416
   &instptr _ instptr.alternative _ $insts + 2;                 08417
   instptr.opcode _ $xecute;                                    08418
   instptr.addr _ sdispptr.dptrun;                              08419
```

```
   &instptr _ $insts;                                                    08420
% trap all signals and put out error disgnostics %                       08421
   ON SIGNAL                                                             08422
      = gaderr:   % address expression error %                           08423
         BEGIN                                                           08424
         *errstr* _ *[MESSAGE]*, " ??";                                  08425
         dismes(2, $errstr);                                             08426
         GOTO errline;                                                   08427
         END;                                                            08428
   ELSE                                                                  08429
      BEGIN                                                              08430
      IF &sysmsg AND sysmsg.L THEN                                       08431
         IF sysmsg.LH < sysmsg.RH                                        08432
            THEN  dismes(2, $"system screwup, invalid value for
            sysmsg")                                                     08433
            ELSE  dismes(2,&sysmsg);                                     08434
      (errline):                                                         08435
      &sysmsg _ 0;                                                       08436
      IF auxinflag THEN auxinterminate();                                08437
      GOTO mainline;                                                     08438
      END;                                                               08439
dpset(dspno, endfil, endfil, endfil);                                    08440
cmdmode _ 0;                                                             08441
IF nlmode                                                                08442
   THEN cmdmode.dnlscmd _ TRUE                                           08443
   ELSE cmdmode.tnlscmd _ TRUE;                                          08444
IF recogmode = mexpert                                                   08445
   THEN cmdmode.l1cmd _ TRUE;                                            08446
(mainline): % main process control loop %                                08447
LOOP                                                                     08448
   BEGIN                                                                 08449
   % process the top entry in the subsystem stack; If the stack is
   empty then we exit the process loop %                                 08450
      IF sbstkx <= 0 THEN EXIT LOOP;                                     08451
      &entry _ $sbstack + sbstkx - $sbentsize;                           08452
      &sdispptr _ entry.sbptr;                                           08453
   % process by the mode of the entry %                                  08454
      CASE entry.sbmode OF                                               08455
         = sbstart:       % initialization %                             08456
            BEGIN                                                        08457
            ptr _ sdispptr.dptinit; % ptr to initialization rule %
                                                                         08458
            entry.sbmode _ sbrun;                                        08459
            % set up some subsystem prompting stuff %                   08460
               sethrld( entry.sbnptr );                                  08461
            % reset command repeat string %                             08462
               *keyrptstr* _ NULL;                                       08463
            END;                                                         08464
         = sbrun: % executing is a subsystem %                           08465
            BEGIN                                                        08466
            IF (entry.sbcount _ entry.sbcount-1 ) = 0                    08467
               THEN entry.sbmode _sbfinish;                              08468
            instptr.addr _ sdispptr.dptrun; % ptr to subsystem
            rule %                                                       08469
            ptr _ $insts;                                                08470
            cueflg _ FALSE;                                              08471
```

```
            fbctl( clearcfl );                                          08472
            % setup for typewriter %                                    08473
               IF nlmode = typewriter THEN                              08474
                  BEGIN                                                 08475
                  FOR count _ 1 UP UNTIL > fedind DO typech(SP);
                                                                        08476
                  typeas( $hrldstr ); % type out the hearald %
                                                                        08477
                  END;                                                  08478
            END;                                                        08479
         = sbfinish:       % termination of the subsystem %             08480
            BEGIN                                                       08481
            ptr _ sdispptr.dptfinish; % ptr to termination rule %
                                                                        08482
            entry.sbmode _ sbpop;                                       08483
            END;                                                        08484
         = sbpop: % pop subsystem %                                     08485
            BEGIN                                                       08486
            sbstkx _ sbstkx - $sbentsize;                               08487
            % set up some subsystem prompting stuff %                  08488
               IF sbstkx > 0 THEN                                       08489
                  BEGIN                                                 08490
                  &entry _ $sbstack + sbstkx - $sbentsize;              08491
                  sethrld( entry.sbnptr );                              08492
                  END;                                                  08493
            % reset command repeat string %                            08494
               *keyrptstr* _ NULL;                                      08495
            REPEAT LOOP;                                                08496
            END;                                                        08497
         = sbrentry:       % reset after leaving subsystem %            08498
            BEGIN                                                       08499
            ptr _ sdispptr.dptrentry;                                   08500
            % reset previous mode %                                     08501
               entry.sbmode _ entry.sbpmode;                            08502
            END;                                                        08503
         ENDCASE                                                        08504
            BEGIN                                                       08505
            dismes( 2,$"interpreter mode screwup, aborted");            08506
            RETURN;                                                     08507
            END;                                                        08508
      % invoke the interpreter to evaluate the rule if there is one %
                                                                        08509
      IF ptr THEN                                                       08510
         BEGIN                                                          08511
         % set up for new command %                                     08512
            pathx _ evalx _ ptrx _ 0;                                   08513
            keyinpstr.L _ 0;                                            08514
            keysaveflag _ basestateflag _  TRUE;                        08515
            IF NOT &tda THEN &tda _ lda();                              08516
            IF nlmode = typewriter THEN                                 08517
               BEGIN                                                    08518
               cspupdate _ &tda;                                        08519
               curmkr _ tda.dacsp;                                      08520
               curmkr[1] _ tda.daccnt;                                  08521
               END                                                      08522
            ELSE cspupdate _ 0;                                         08523
```

```
            cspvs _ tda.davspec;  cspvs[1] _ tda.davspc2;          08524
            cspusqcod _ tda.dausqcod;                              08525
            cspcacode _ tda.dacacode;                              08526
          % invoke the interpreter %                               08527
            cmdinterp( $pathstk+3, $ptr );                         08528
          % reset after successful command completion %            08529
            (cmdend):                                              08530
            cmdfinish();                                           08531
            IF analyzing THEN                                      08532
               BEGIN                                               08533
               !time();                                            08534
               notetime(103B %command end%, r1);                   08535
               analtime _ TRUE;                                    08536
               END;                                                08537
            IF msjfn THEN  %monitoring commands%                   08538
               BEGIN  %note end of command and check last time load
               average entered%                                    08539
               msrecord(mdone, 0, 0);                              08540
               IF mnaproc AND SKIP !getab(mnaproc) THEN            09790
                  BEGIN                                            09791
                  r3 _ fone;                                       09792
                  !FADRM r3,msumrcnt;                              09793
                  !FDVR r3,r1;                                     09794
                  !FADRM r3,msumrap;                               09795
                  END;                                             09796
               gtadcall();                                         08541
               IF r1 - ldavgtime >= ldavgsec THEN                  08542
                  BEGIN  %enter a new one%                         08543
                  ldavgtime _ r1;                                  08544
                  *loadstr* _ "    ";                              08545
                  IF SKIP !getab(lavtabad) THEN                    08546
                     BEGIN                                         08547
                     loadavg _ r1;                                 08548
                     IF SKIP !flout($loadstr+chbmty, loadavg, 0) THEN
                                                                   08549
                        loadstr.L _ slngth($loadstr+chbmty, r1);
                                                                   08550
                     IF mnaproc AND SKIP !getab(mdshare) THEN      09797
                        BEGIN                                      09798
                        !FMPR r1,msumrap;                          09799
                        !FDVR r1,msumrcnt;                         09800
                        %do something with r1%                     09801
                        !LSH r1,-21; %get xnnnnx,,xxxxxx%          09802
                        !ANDI r1,7777B;                            09803
                        msumrap _ msumrcnt _ 0;                    09804
                        END                                        09805
                     ELSE r1 _ 0;                                  09806
                     msrecord(mloadavg, $loadstr, r1);             08551
                     END;                                          08552
                  END;                                             08553
               END;                                                08554
            END;                                                   08555
         END;                                                      08556
      % if we fall out of the main loop, then the subsystem stack must
      be empty and we are exiting the system %                     08557
         RETURN;                                                   08558
```

```
   END.
                                                                    08559
(cmdfinish) PROCEDURE;                                               0535
   REF tda;                                                         03407
   LOCAL srr, frr, stid, cc;  REF frr, srr;                          0536
   LOCAL STRING locstr[200];                                         0537
   IF cspupdate THEN                                                 0538
      BEGIN                                                          0539
      &tda _ cspupdate;                                              0540
      stid _ tda.dacsp;                                              0541
      cc _ tda.daccnt;                                               0542
      % set so ^O will clear output buffer %                        09808
         rubnocob _ FALSE;                                          09809
      %update statement return ring%                                 0543
         &frr _ tda.dalink;  %get address of file return ring%       0544
         IF NOT frr.frhexis THEN                                     0545
            err($"Illegal file return ring detected in cmdfinish");
                                                                     0546
         %get frr  entry address%                                    0547
            &frr _ &frr + frrhlen + (frrelen*frr.frhtop);            0548
         IF frr.frexis AND NOT tda.daempty AND tda.dacsp NOT= endfil
         THEN %update srr%                                           0549
            BEGIN                                                    0550
            %get address of statement return ring%                   0551
               &srr _ frr.frsrring;                                  0552
            %update old position and viewspecs on ring%              0553
               storesrring(&srr, 0, tda.dacsp, tda.daccnt,
               tda.davspec, tda.davspc2);                            0554
                  %user may have changed viewspecs%                  0555
            END;                                                     0556
      IF curmkr.stfile NOT= tda.dacsp.stfile THEN  %changing files,
      push file return ring%                                         0557
         BEGIN                                                       0558
         %get name of new file%                                      0559
            *locstr* _ NULL;                                         0560
            filnam(curmkr.stfile, $locstr);                          0561
         %push new file name on ring%                                0562
            pushfrring(tda.dalink, $locstr, curmkr.stfile);          0563
            readfrring(tda.dalink, 0 : &srr);                        0564
            IF usesrr THEN  %jump file return -- copy usesrr to new
            srr%                                                     0565
               BEGIN                                                 0566
               copysrring(usesrr, &srr);                             0567
               END;                                                  0569
         %put out "modified" message if necessary%                   0570
            IF [flntadr(curmkr.stfile)].fllock THEN                  0571
               lockmes(curmkr.stfile)                                0572
            ELSE dismes(2, $locstr);  %show user new file name%      0573
         %close files no longer used in display areas%               0574
            tda.dacsp _ curmkr;                                      0575
            tda.daccnt _                                             0576
               IF nlmode = fulldisplay THEN 1 ELSE curmkr[1];        06334
            tda.daempty _ FALSE;                                     0577
            freflnt(); %close files no longer needed%                0578
         END;                                                        0579
      tda.dacsp _ curmkr;                                            0580
```

```
      tda.daccnt _                                                    06335
         IF nlmode = fulldisplay THEN 1 ELSE curmkr[1];               06336
      tda.daempty _ FALSE;                                             0582
      %update viewspecs%                                               0583
         IF nlmode # fulldisplay THEN                                 08326
            BEGIN  %pvs fields will be checked and set by recred in
            DNLS%                                                     08327
            tda.dapvs _ tda.davspec;                                  08330
            tda.dapvs2 _ tda.davspc2;                                 08331
            END;                                                      08328
         tda.davspec _ cspvs;                                         0584
         tda.davspc2 _ cspvs[1];                                      0585
         tda.dacacode _ cspcacode;                                    0586
         tda.dausqcod _ cspusqcod;                                    0587
      %push new position and viewspecs onto statement return ring%
                                                                      0588
         IF (NOT (usesrr := 0)) AND (stid NOT= curmkr OR cc NOT=
         curmkr[1]) THEN                                              0589
            pushsrring(&srr, tda.dacsp, tda.daccnt, tda.davspec,
            tda.davspc2);                                             0590
                                                                      0591
      END;                                                            0592
   IF nlmode = fulldisplay THEN                                       0593
      BEGIN                                                           0594
      % recreate the display %                                        0595
         recred();                                                    0596
         cdtype _ dspno; % shut off recred until dpset is called
         again %                                                      0597
      IF lplitreset THEN                                             07136
         BEGIN                                                       07137
         litline _ lplitline;                                        07138
         litreset _ FALSE;                                           07139
         litapflag _ TRUE;                                           07140
         rstlit();                                                   07141
         lplitreset _ FALSE;                                         07142
         END;                                                        07143
      END;                                                            0598
   noclrall _ FALSE;  %set TRUE by xsubstitute to keep message in tty
   window; checked by clearda%                                        09807
   bugtrap();  %to help find whatever troublesome bugs are around at
   the time -- normally just a return%                                06530
   RETURN;                                                            0599
   END.                                                               0600
                                                                      0601
(bugtrap) PROC; %set traps for elusive bugs%                          06531
   %          --formerly named ckrrings--
   Commented out by KJM 28-JAN-77 because the original return ring
   errors never seem to happen.  The call to this procedure left in
   to facilitate short-term bug trapping checks.
                                                                      09827
   LOCAL da, end, frr, srr, frrend;                                   06535
   REF da, frr, srr;                                                  06536
   end _ (&da _ $dpyarea) + dal*dacnt;                                06537
   DO IF da.daexis AND NOT da.daempty THEN                            06538
      BEGIN                                                           06540
      IF NOT (&frr _ da.dalink) THEN                                  06542
```

```
            werr($"return ring error: dalink empty");                    06543
         IF NOT frr.frhexis THEN                                         06544
            werr($"return ring error: FRR empty");                       06551
         frrend _ &frr + frr.frhlast*frrelen + frrhlen;                  06556
         FOR &frr _ &frr + frrhlen UP frrelen UNTIL >=frrend DO          06545
            IF frr.frexis THEN                                           06546
               BEGIN                                                     06553
               IF NOT (&srr _ frr.frsrring) THEN                         06547
                  werr($"return ring error: FRSRING field empty");       06554
               IF NOT srr.srhexis THEN                                   06548
                  werr($"return ring error: SRR empty");                 06555
               END;                                                      06552
         END                                                             06541
      UNTIL (&da _ &da + dal) >= end;                                    06539
      %                                                                  09828
      RETURN;                                                            06532
      END.                                                               06533
                                                                         06534
   (sethrld) PROC( % sets up command hearald based upon subsystem name %
                                                                         0602
      % FORMAL ARGUMENTS %                                               0603
         strptr);       % ptr to hearald name string %                   0604
      REF strptr;                                                        0605
      %------------%                                                     0606
      % save subsystem name away in global string ssysname %            0607
         *ssysname* _ *strptr*;                                          0608
      CASE hrldmode OF                                                   0611
         = onechar:     % single char hearald %                         0612
            *hrldstr* _ "* ";                                            0613
         = multchar:    % multiple char hearald %                       0614
            *hrldstr* _ *strptr* [1 TO MIN(hrldsize, strptr.L,
            hrldstr.M-1)], SP;                                           0615
         ENDCASE;                                                        0616
      IF nlmode = fulldisplay                                            0609
         THEN                                                            0617
            BEGIN       % display subsystem name %                       0618
            dsubsys( &strptr );                                          0619
            END;                                                         0620
      RETURN;                                                            0621
      END.

                                                                         0622
% COMMAND INTERPRETER %                                                 0623
   % MAIN CONTROL ROUTINES %                                            0624
      (cmdinterp) PROCEDURE(                                           08560
         % FORMAL ARGUMENTS %                                          08561
            resultptr,  % pointer to the result record %               08562
            argptr );   %pointer to a function state record containing a
            pointer to a node in the grammar %                       08563
         % NORMAL RETURNS %                                           08564
            % 1) pointer to a function state record of the result,  This
            pointer is set to 0 if the parse fails. %                08565
         % ABNORMAL RETURNS %                                         08566
            % abnormal returns are accomplished via SIGNALS, generated
            for the following conditions %                           08567
               % 1) interperr -- an interpreter failure is detected %
                                                                     08568
```

```
        % 2) cmddelete -- the command is aborted due to a command
        delete char. %                                          08569
    LOCAL % VARIABLES %                                         08570
       tempptr,     % temporary pointer, working value %        08571
       instptr,     % ptr to interpretive text instruction %    08572
       curptr,      % ptr to current path stack entry %         08573
       nextptr,     % ptr to next path stack entry %            08574
       firstptr,    % ptr to first path stack entry %           08575
       stateptr,    % ptr to function state record %            08576
       pathptr,     % ptr to start of selection path %          08577
                    % = 0 if selection just completed %         08578
       function;    % address of processing function %          08579
    LOCAL STRING locstr[100];                                   08580
    REF % VARIABLES %                                           08581
       pathptr,                                                 08582
       tempptr,                                                 08583
       instptr,                                                 08584
       function,                                                08585
       curptr,                                                  08586
       nextptr,                                                 08587
       firstptr,                                                08588
       stateptr,                                                08589
       resultptr,                                               08590
       argptr;                                                  08591
    %------------------------%                                  08592
    % trap any state changing signals %                         08593
       ON SIGNAL                                                08594
          = popstate:                                           08595
            BEGIN                                               08596
            % curptr points to a selection function frame.  We
            want to back out of this frame, to the beginning of
            the previous frame if there is one %                08597
               WHILE &curptr >= &pathptr DO                     08598
                  % back out of current frame %                 08599
                  BEGIN                                         08600
                  % check for backup in/into a selection function
                  %                                             08601
                     IF testselect( &curptr ) THEN              08602
                        BEGIN                                   08603
                        &nextptr _ &curptr + $totalrecsize;     08604
                        &instptr _ curptr.begnodeptr;           08605
                        &pathptr _ curptr.markptr;              08606
                        pathx _ &nextptr - $pathstk;            08607
                        GOTO parseit;                           08608
                        END;                                    08609
                  % $call processing function in "cleanup" mode %
                                                                08610
                     IF (&function _ curptr.pfunction) # 0  THEN
                                                                08611
                        function( &curptr+$pathrecsize, cleanup );
                                                                08612
                  % back curptr down to previous frame %        08613
                     &curptr _ &curptr - $totalrecsize;         08614
                  END;                                          08615
            % curptr now points to the frame preceeding the top
            one (if one exists).  set up local interpreter
```

```
variables and the global state variables and resume
the parse at the top position of the current path in
the grammar %                                          08616
LOOP                                                   08617
   BEGIN                                               08618
   % reset the prompting flag %                        08619
      cueflg _ FALSE;                                  08620
   % collapse the path stack down to the last
   selection function %                                08621
      &nextptr _ &curptr _ &curptr + $totalrecsize;
                                                       08622
      pathx _ &nextptr - $pathstk;                     08623
      ptrx _ curptr.ptrxsav;                           08624
      evalx _ curptr.evalxsav;                         08625
   % let the signal propagate if we have reached the
   bottom of the path stack %                          08626
      IF &curptr <= &firstptr THEN EXIT LOOP;          08627
   % back the path stack up to the beginning of the
   previous selection %                                08628
      &curptr _ &curptr - $totalrecsize;               08629
      &pathptr _ curptr.markptr;                       08630
   IF msjfn THEN  %monitoring commands%                08631
      msrecord(mpopstate, 0, 0);                       08632
   % curptr points to a selection function frame.  We
   want to back out of this frame, to the beginning of
   the previous frame if there is one %                08633
      WHILE &curptr >= &pathptr DO                     08634
         % back out of current frame %                 08635
         BEGIN                                         08636
         % check for backup in/into a selection
         function %                                    08637
            IF testselect( &curptr ) THEN              08638
               BEGIN                                   08639
               &nextptr _ &curptr + $totalrecsize;
                                                       08640
               &instptr _ curptr.begnodeptr;           08641
               &pathptr _ curptr.markptr;              08642
               pathx _ &nextptr - $pathstk;            08643
               GOTO parseit;                           08644
               END;                                    08645
         % $call processing function in "cleanup" mode
         %                                             08646
            IF (&function _ curptr.pfunction) # 0
            THEN                                       08647
               BEGIN                                   08648
               IF msjfn AND &function = $keywrec THEN
                                                       08649
                  msrecord(mcompopstate,
                  [&curptr+$pathrecsize+1], 0);        08650
               function( &curptr+$pathrecsize, cleanup
               );                                      08651
               END;                                    08652
         % back curptr down to previous frame %         08653
            &curptr _ &curptr - $totalrecsize;         08654
         END;                                          08655
   %check if ignoring level adjust or viewspecs %
```

```
                                                                    08656
            IF nolevadj AND [pathptr.curnodeptr].opcode =
            $levadj                                                 08657
               THEN REPEAT LOOP;                                    08658
            IF novspec AND [pathptr.curnodeptr].opcode =
            $vwspecs                                                08659
               THEN REPEAT LOOP;                                    08660
         % set up to resume execution %                             08661
            % collapse the path stack %                             08662
               &nextptr _ &curptr _ &curptr + $totalrecsize;
                                                                    08663
               pathx _ &nextptr - $pathstk;                         08664
            &instptr _ curptr.begnodeptr;                           08665
            &pathptr _ 0;                                           08666
            ptrx _ curptr.ptrxsav;                                  08667
            evalx _ curptr.evalxsav;                                08668
            curptr.pmode _ parsing;                                 08669
            fbctl( fbpop );                                         08670
            GOTO parseit;                                           08671
         END;                                                       08672
      END;                                                          08673
= cutpathstk:                                                       08674
      BEGIN % cut stack back to frame indicated by cutstop %
                                                                    08675
      IF cutstop > 0 THEN                                           08676
         WHILE &curptr > cutstop DO                                 08677
            % back out of current frame %                           08678
            BEGIN                                                   08679
            % $call processing function in "cleanup" mode %
                                                                    08680
               IF (&function _ curptr.pfunction) # 0  THEN
                                                                    08681
                  function( &curptr+$pathrecsize, cleanup );
                                                                    08682
            % back curptr down to previous frame %                  08683
               &curptr _ &curptr - $totalrecsize;                   08684
            END;                                                    08685
      % curptr now points to the frame specified by cutstop.
      set up local interpreter variables and the global
      state variables and resume the parse at the top
      position of the current path in the grammar %                 08686
      LOOP                                                          08687
         BEGIN                                                      08688
         % reset the prompting flag %                               08689
            cueflg _ FALSE;                                         08690
         % collapse the path stack down to the last
         selection function %                                       08691
            &nextptr _ &curptr _ &curptr + $totalrecsize;
                                                                    08692
            pathx _ &nextptr - $pathstk;                            08693
            ptrx _ curptr.ptrxsav;                                  08694
            evalx _ curptr.evalxsav;                                08695
         % let the signal propagate if we have reached the
         bottom of the path stack %                                 08696
            IF &curptr <= &firstptr THEN EXIT LOOP;                 08697
         % back the path stack up to the beginning of the
```

```
                previous frame %                                    08698
                   &curptr _ &curptr - $totalrecsize;               08699
                   &pathptr _ curptr.markptr;                       08700
                %check if ignoring level adjust or viewspecs %
                                                                    08701
                   IF nolevadj AND [pathptr.curnodeptr].opcode =
                   $levadj                                          08702
                      THEN REPEAT LOOP;                             08703
                   IF novspec AND [pathptr.curnodeptr].opcode =
                   $vwspecs                                         08704
                      THEN REPEAT LOOP;                             08705
                % set up to resume execution %                      08706
                   % collapse the path stack %                      08707
                      &nextptr _ &curptr _ &curptr + $totalrecsize;
                                                                    08708
                      pathx _ &nextptr - $pathstk;                  08709
                   &instptr _ curptr.begnodeptr;                    08710
                   &pathptr _ 0;                                    08711
                   ptrx _ curptr.ptrxsav;                           08712
                   evalx _ curptr.evalxsav;                         08713
                   curptr.pmode _ parsing;                          08714
                   %fbctl( fbpop ); not necessary and causes
                   problems for HELP (extra "_") , therefore
                   removed%                                         08715
                   GOTO parseit;                                    08716
                END;                                                08717
             END;                                                   08718
          ELSE % all other signals must be error/command delete
          signals %                                                 08719
             BEGIN                                                  08720
             (abort):                                               08721
             IF msjfn THEN   %monitoring commands%                  08722
                %note command delete and errors%                    08723
                IF sysgnl = cmdelete THEN                           08724
                   msrecord(mcomdel, 0, 0)                          08725
                ELSE                                                08726
                   msrecord(merrsig, 0, 0);                         08727
             WHILE &curptr >= &firstptr DO                          08728
                BEGIN                                               08729
                % $call processing function in "cleanup" mode %
                                                                    08730
                   IF (&function _ curptr.pfunction) # 0            08731
                      THEN function( &curptr+$pathrecsize, cleanup
                      );                                            08732
                % cleanup to previous path stack entry and collapse
                the stack %                                         08733
                   &curptr _ &curptr - $totalrecsize;               08734
                END;                                                08735
             % reset the prompting flag %                           08736
                cueflg _ FALSE;                                     08737
             % reset completion code %                              08738
                complcode _ 1;                                      08739
             END;                                                   08740
       % initialize interpreter variables %                        08741
          &instptr _ argptr;                                        08742
          &curptr _ &nextptr _ &firstptr _ $pathstk + pathx;        08743
```

```
    &pathptr _ 0;                                                   08744
% initialize local variables %                                      08745
    curptr.pmode _ parsing;                                         08746
(parseit):                                                          08747
% continue parsing as long as we are doingit %                      08748
    WHILE &curptr DO                                                08749
        BEGIN                                                       08750
        CASE curptr.pmode OF % process according to type of parse
        %                                                           08751
            = parsing:              % normal parsing mode %        08752
                BEGIN                                               08753
                % set up a new path stack entry %                   08754
                    &curptr _ &nextptr;                             08755
                    curptr.begnodeptr _ &instptr;                   08756
                    CASE &nextptr _ &nextptr + $totalrecsize OF
                                                                    08757
                        >= $pathstk + $pssize:                      08758
                            SIGNAL (interperr, $"Path Stack
                            Overflowed");                           08759
                        > $pathstk + pathx:                         08760
                            BEGIN                                   08761
                            % no keyword recognition yet %          08762
                                kwrstate _ 0;                       08763
                            pathx _ &nextptr - $pathstk;            08764
                            END;                                    08765
                    ENDCASE;                                        08766
                    % initialize frame values which are not altered
                    after backup %                                  08767
                        curptr.ptrxsav _ ptrx;                      08768
                        curptr.evalxsav _ evalx;                    08769
                    (resume):  % resume here after backup %         08770
                    % set pathptr to curptr if not already set
                    do not update pathptr if next inst is a STORE %
                                                                    08771
                        IF NOT &pathptr AND instptr.opcode # $store
                                                                    08772
                            THEN &pathptr _ &curptr;                08773
                    % bump frame counter (10 bits resolution) %
                                                                    08774
                        framecounter _ (framecounter+1) .A 1777B;
                                                                    08775
                    % initialize the path record %                 08776
                        curptr.curnodeptr _ &instptr;               08777
                        curptr.pmode _ parsing;                     08778
                        curptr.evalmod _ $unknown;                  08779
                        curptr.fcounter _ framecounter;             08780
                        curptr.pfunction _ 0;                       08781
                        curptr.markptr _                            08782
                            IF &pathptr THEN &pathptr               08783
                            ELSE [&curptr - $totalrecsize].markptr;
                                                                    08784
                    % set function state ptr %                     08785
                        &stateptr _ &curptr + $pathrecsize;         08786
                    % evaluate the upcoming node %                 08787
                        evaler( &curptr );                          08788
                    END;                                            08789
```

```
= cleanup:              % termination of a command %
                                                        08790
   BEGIN                                                08791
   % save current keyword save string for possible
   upcoming repeat command function %                   08792
      *keyrptstr* _ *keyinpstr*;                        08793
   % backup in the command as far as necessary
   (depending on the type of temination %               08794
      &tempptr _ IF complcode = 1                       08795
         THEN &firstptr                                 08796
         ELSE setbackup(&firstptr, &curptr);            08797
      WHILE &curptr >= &tempptr DO                      08798
         BEGIN                                          08799
         % $call the execution function in cleanup
         mode if one exists %                           08800
            IF (&function _ curptr.pfunction ) # 0
                                                        08801
               THEN function ( &curptr+$pathrecsize,
               cleanup);                                08802
         &curptr _ &curptr - $totalrecsize;             08803
         END;                                           08804
   % terminate processing, prepare for rpt or insert
   if required %                                        08805
      CASE complcode OF                                 08806
         = 1: % normal CA %                             08807
            RETURN;                                     08808
         = 2: % insert statement mode %                 08809
            BEGIN                                       08810
            &curptr _ &tempptr;                         08811
            &instptr _ $zinsstatement;                  08812
            curptr.begnodeptr _ &instptr;               08813
            IF msjfn THEN  %monitoring commands%        08814
               msrecord(minsert, 0, 0);  %note cntl-E%
                                                        08815
            END;                                        08816
         = 3: % repeat command mode %                   08817
            BEGIN                                       08818
            &curptr _ &tempptr;                         08819
            &instptr _ curptr.begnodeptr;               08820
            IF nlmode = fulldisplay THEN cfldsp();
                                                        08821
            IF msjfn THEN  %monitoring commands%        08822
               msrecord(mrepeat, 0, 0);  %note cntl-B%
                                                        08823
            END;                                        08824
         ENDCASE err(notyet);                           08825
   % prepare to resume execution %                     08826
      ptrx _ curptr.ptrxsav;                            08827
      evalx _ curptr.evalxsav;                          08828
      &nextptr _ &curptr + $totalrecsize;               08829
      &pathptr _ 0;                                     08830
      cueflg _ FALSE;                                   08831
   cmdfinish();                                         08832
   IF NOT &tda THEN &tda _ lda();                       08833
   cspupdate _ IF nlmode = typewriter THEN &tda ELSE
   0;                                                   08834
```

```
        % following two intructions nopped by kev 6/4/74 %
                                                              08835
           % curmkr _ tda.dacsp; %                            08836
           % curmkr[1] _ tda.daccnt; %                        08837
        cspvs _ tda.davspec;  cspvs[1] _ tda.davspc2;         08838
        cspusqcod _ tda.dausqcod;                             08839
        cspcacode _ tda.dacacode;                             08840
        GOTO resume;                                          08841
        END;                                                  08842
      = pnext:                                                08843
        BEGIN                                                 08844
        basestateflag _ FALSE; % no longer in base command
        state %                                               08845
        &instptr _ curptr.curnodeptr;                         08846
        % reset pathptr if last instruction was a selection
        function %                                            08847
           IF instptr.opcode IN [$keyop, $levadj]             08848
              THEN                                            08849
                 BEGIN                                        08850
                 &pathptr _ 0;                                08851
                 lastsel _ instptr.opcode;                    08852
                 END;                                         08853
        % stop collecting keyword input strings (for
        possible command repeat) if we just finished a
        non-keyword recognizer %                              08854
           IF instptr.opcode IN [$confirm, $call]             08855
              THEN keysaveflag _ FALSE;                       08856
        CASE &instptr _ instptr.nsuccessor OF                 08857
           = 0:    % null nsuccessor %                        08858
              IF ptrx > 0                                     08859
                 THEN                                         08860
                    BEGIN                                     08861
                    &instptr _ ptrstk[ ptrx _ ptrx-1 ];
                                                              08862
                    REPEAT CASE;                              08863
                    END                                       08864
                 ELSE                                         08865
                    REPEAT CASE 2 (cleanup);                  08866
        ENDCASE;                                              08867
        curptr.pmode _ parsing;                               08868
        END;                                                  08869
      = backup:              % backup for repeat %            08870
        BEGIN                                                 08871
        &nextptr _ &curptr;                                   08872
        LOOP                                                  08873
           BEGIN                                              08874
           &instptr _ curptr.curnodeptr;                      08875
           % $call the execution function in backup mode if
           one exists %                                       08876
              IF (&function _ curptr.pfunction ) # 0          08877
                 THEN function ( &curptr+$pathrecsize,
                 backup);                                     08878
           % try to find an alternative to the current inst
           %                                                  08879
              IF curptr.evalmod # $parallel                   08880
                 THEN                                         08881
```

```
                        BEGIN                                   08882
                        CASE &instptr _ instptr.alternative OF
                                                                08883
                           = 0:                                 08884
                              IF ptrx > curptr.ptrxsav THEN
                                                                08885
                                 BEGIN                          08886
                                 &instptr _ ptrstk[ ptrx _
                                 ptrx-1];                       08887
                                 REPEAT CASE;                   08888
                                 END;                           08889
                              ENDCASE                           08890
                                 EXIT LOOP;                     08891
                        END                                     08892
                     ELSE                                       08893
                        BEGIN                                   08894
                        % check to see if we are in a perform
                        loop %                                  08895
                           &instptr _ curptr.begnodeptr;        08896
                           IF &instptr = instptr.alternative
                           THEN EXIT LOOP;                      08897
                        END;                                    08898
                  % reset state counters %                      08899
                     ptrx _ curptr.ptrxsav;                     08900
                     evalx _ curptr.evalxsav;                   08901
                  % current inst. has no alternative, so back up
                  path stack until we find a alternative.  If none
                  is found, then the command is terminated %    08902
                     IF (&curptr _ &curptr - $totalrecsize) <
                     &firstptr                                  08903
                        THEN                                    08904
                           RETURN (FALSE); % parse failed %     08905
                  END;                                          08906
                  % setup to resume execution with current path
                  stack entry %                                 08907
                     cueflg _ FALSE;                            08908
                     &pathptr _ 0;                              08909
                     &nextptr _ &curptr + $totalrecsize;        08910
                     % reset path stack index %                 08911
                        pathx _ &nextptr - $pathstk;            08912
                     % reset eval stack index %                 08913
                        evalx _ curptr.evalxsav;                08914
                     % reset feedback %                         08915
                        IF nlmode = fulldisplay THEN cfldsp();
                                                                08916
                     % resume execution %                       08917
                        GOTO resume;                            08918
                                                                08919
               END;                                             08920
            = popselect:  % backup to inside of selection fuction
            %                                                   08921
               BEGIN                                            08922
               % invoke the selection processor in pop mode.  We
               pass only the resultptr and parsemode.  The rest of
               the work will be done in the selection processor %
                                                                08923
```

```
                    xselect( &curptr+$pathrecsize, popselect); 08924
                 % push the address of the result record onto the
                 eval stack %                                   08925
                    xpush( &curptr+$pathrecsize );              08926
                 curptr.pmode _ pnext;                          08927
                 END;                                           08928
           ENDCASE SIGNAL (interperr, $"Unrecognized Parse Mode");
                                                                08929
           END;                                                 08930
      % we've screwed up if we get to here. %                   08931
        SIGNAL (interperr, $"Parser screwed up");               08932
      END.
                                                                08933
   (setbackup) PROC( % defines backup point after command completion
   for repeat or insert mode command termination %               0921
      % FORMAL ARGUMENTS %                                      0922
        firstptr,    % ptr to first path stack entry %          0923
        curptr);     % ptr to current path stack entry %        0924
      % RETURNS %                                               0925
        % ptr to path stack to which backup is to proceed %     0926
      LOCAL % VARIABLES %                                       0927
        pathptr,     % ptr to head of path stack frame %        0928
        ptr;         % ptr to path stack entry %                0929
      REF ptr, firstptr, curptr, pathptr;                       0930
      %------------%                                           0931
      CASE complcode OF                                         0932
        = 2:         % insert stmt %                           0933
           BEGIN                                                0934
           % check to make sure that the current system is the
           nlseditor, if not SIGNAL cmdelete which will cleanup as
           per CA character %                                   0935
              IF [$sbstack + sbstkx - $sbentsize].sbptr # $nlseditor
                                                                0936
                 THEN SIGNAL( cmdelete );                       0937
           &ptr _ &firstptr;                                    0938
           END;                                                 0939
        = 3:         % rpt char %                              0940
           BEGIN                                                0941
           % rumble down path stack frames looking for one not
           beginning with a keyword recognition function %      0942
              &pathptr _ 0;                                     0943
              FOR &ptr _ &firstptr UP $totalrecsize UNTIL > &curptr
              DO                                                0944
                 BEGIN                                          0945
                 IF ptr.markptr # &pathptr THEN                 0946
                    &pathptr _ &ptr;                            0947
                 CASE [ptr.curnodeptr].opcode OF                0948
                    IN [$confirm, $call]:                       0949
                       RETURN( &pathptr );                      0950
                    ENDCASE;                                    0951
                 END;                                           0952
           % if we fall through to here, then the command can't be
           repeated, so signal the cmdelete %                   0953
              SIGNAL (cmdelete);                                0954
           END;                                                 0955
        ENDCASE err(notyet);                                    0956
```

```
    RETURN( &ptr );                                                    0957
    END.
                                                                       0958
(sleuth) PROCEDURE(                                                    02626
    % sleuth examines the current alternatives for the interpreter
    and sets up the action records to record what happens when one
    of the trigger characters is recognized  %                         02627
    % FORMAL ARGUMENTS %                                               02628
       psptr,       % ptr to the current path stack record %         02629
       optptr,      % ptr to opt action record %                     02630
       captr,       % ptr to ca action record %                      02631
       defaultptr);% ptr to default action record %                   02632
    % NORMAL RETURNS %                                                02633
       % The action records for the opt action, ca action, and
       default action are set up and completely filled in. %           02634
       % 1): the interpretation mode is returned  %                   02635
          % = $parallel: $parallel recognition mode: %                02636
          % = $serial: $serial function execution mode %              02637
       % 2): a count of the number of alternative execution paths %
                                                                       02638
    % ABNORMAL RETURNS %                                              02639
       % a signal is generated whenever an ambiguous construction
       in the grammar is detercted. %                                  02640
    LOCAL % VARIABLES %                                                02641
       op, % opcode of instruction %                                   02642
       i,           % loop index variable %                           02643
       brcount,     % number of branches in recognition segence %
                                                                       02644
       returnval,   % function return value %                         02645
       headptr,     % ptr to head of alternative branch %            02646
       resetheader,% flag for resetting headptr %                      02647
       instptr,     % ptr to instruction %                           02648
       lptrx,       % index of next location in lptrstk %            02649
       trivial,     % flag to prevent "trivial alternative"%         03304
       % lptrstk SHOULD BE ptrssize (in PDATA) WORDS LONG %           03760
          lptrstk[40]; % work stack for tree chasing %                 02650
    REF % VARIABLES %                                                  02651
       headptr,     % ptr to head of nsuccessor list %               02652
       instptr,     % ptr to instruction %                           02653
       psptr,       % ptr to the current path stack record %         02654
       optptr,      % ptr to opt action record %                     02655
       captr,       % ptr to $pca action record %                    02656
       defaultptr; % ptr to default action ptr %                      02657
    %-----------------%                                               02658
    % initialize all fields of action records to default values %
                                                                       02659
       optptr.propcode _ optptr.fbstrptr _ optptr.insptr _ 0;          02660
       captr.propcode _ captr.fbstrptr _ captr.insptr _ 0;             02661
       defaultptr.propcode _ defaultptr.fbstrptr _
       defaultptr.insptr _ 0;                                          02662
    trivial _ TRUE;                                                    03305
    % initialize local variables %                                    02663
       &headptr _ &instptr _ psptr.curnodeptr;                         02664
       resetheader _ FALSE;                                            02665
       lptrx _ 0;                                                      02666
       returnval _ $parallel;                                          02667
```

```
   brcount _ 0;                                                    02668
% we now set up actions to be accompilshed whenever a trigger
character is encountered .  The characters currently recognized
as trigger characters are :                                       02669
   1) CA character-- may be a bug selection.                      02670
   2) option char-- indicated what to do when an $option is
   typed.                                                         02671
   3) any other char: --this is the default action for the type
   of recognition. %                                              02672
   % the action function is set to 0 if it is not permitted %
                                                                  02673
% in order to set up all actions, we must check all
alternatives to the current instruction, as they will be
recognized in $parallel %                                         02674
   WHILE &instptr # 0 DO                                          02675
      BEGIN                                                       02676
      CASE op _ instptr.opcode OF                                 02677
         = $keyop:     % keyword recognition %                    02678
          BEGIN                                                   02679
          (keywdlabel):                                           07152
          BUMP brcount;                                           02680
          CASE defaultptr.propcode OF                             02681
             = 0:    % not yet defined %                          02682
                BEGIN                                             02683
                defaultptr.propcode _ op;                         02684
                defaultptr.insptr _ &headptr;                     02685
                defaultptr.fbstrptr _ (IF psptr.curnodeptr =
                psptr.begnodeptr THEN $"C:" ELSE 0);              07151
                END;                                              02687
             # op:                                                02688
                SIGNAL (interperr, $"Ambiguous Grammar --
                multiple default actions ");                      02689
          ENDCASE;                                                02690
          END;                                                    02691
         = $option,                                               02692
         = $anyof:     % optional constructs %                    02693
          BEGIN                                                   02694
          (optlabel):                                             07153
          CASE optptr.propcode OF                                 02695
             = 0:    % not yet defined %                          02696
                BEGIN                                             02697
                BUMP brcount;                                     02698
                optptr.propcode _ op;                             02699
                optptr.insptr _ &headptr;                         02700
                optptr.fbstrptr _ (IF psptr.curnodeptr =
                psptr.begnodeptr THEN $"[**]:" ELSE 0);           02701
                END;                                              02702
             # op:                                                02703
                SIGNAL (interperr, $"Ambiguous Grammar --
                multiple optional actions ");                     02704
          ENDCASE;                                                02705
          resetheader _ TRUE;                                     02706
          REPEAT CASE (-1);                                       02707
          END;                                                    02708
         = $levadj:    % level adjust %                           02709
          BEGIN                                                   02710
```

```
                (levlabel):                                          07154
                CASE defaultptr.propcode OF                          02711
                   = 0:    % not yet defined %                       02712
                      BEGIN                                          02713
                      BUMP brcount;                                  02714
                      defaultptr.propcode _ op;                      02715
                      defaultptr.insptr _ &headptr;                  02716
                      defaultptr.fbstrptr _ (IF psptr.curnodeptr =
                      psptr.begnodeptr THEN $"L:" ELSE 0);           02717
                      END;                                           02718
                   # op:                                             02719
                      SIGNAL (interperr, $"Ambiguous Grammar --
                      multiple default actions ");                   02720
                ENDCASE;                                             02721
                CASE captr.propcode OF                               02722
                   = 0:    % not yet defined %                       02723
                      BEGIN                                          02724
                      captr.propcode _ op;                           02725
                      captr.insptr _ &headptr;                       02726
                      captr.fbstrptr _ (IF psptr.curnodeptr =
                      psptr.begnodeptr THEN $"L:" ELSE 0);           02727
                      END;                                           02728
                   # op:                                             02729
                      SIGNAL (interperr, $"Ambiguous Grammar --
                      multiple $pca actions ");                      02730
                ENDCASE;                                             02731
                END;                                                 02732
             = $vwspecs:    % viewspecs %                            02733
                BEGIN                                                02734
                (vslabel):                                           07155
                CASE defaultptr.propcode OF                          02735
                   = 0:    % not yet defined %                       02736
                      BEGIN                                          02737
                      BUMP brcount;                                  02738
                      defaultptr.propcode _ op;                      02739
                      defaultptr.insptr _ &headptr;                  02740
                      defaultptr.fbstrptr _ (IF psptr.curnodeptr =
                      psptr.begnodeptr THEN $"V:" ELSE 0);           02741
                      END;                                           02742
                   # op:                                             02743
                      SIGNAL (interperr, $"Ambiguous Grammar --
                      multiple default actions ");                   02744
                ENDCASE;                                             02745
                CASE captr.propcode OF                               02746
                   = 0:    % not yet defined %                       02747
                      BEGIN                                          02748
                      captr.propcode _ op;                           02749
                      captr.insptr _ &headptr;                       02750
                      captr.fbstrptr _ (IF psptr.curnodeptr =
                      psptr.begnodeptr THEN $"V:" ELSE 0);           02751
                      END;                                           02752
                   # op:                                             02753
                      SIGNAL (interperr, $"Ambiguous Grammar --
                      multiple $pca actions ");                      02754
                ENDCASE;                                             02755
                END;                                                 02756
```

```
        = $confirm:   % command confirmations %                    02757
          (conlabel):                                              07156
          CASE captr.propcode OF                                   02758
             = 0:    % not yet defined %                           02759
                BEGIN                                              02760
                BUMP brcount;                                      02761
                captr.propcode _ op;                               02762
                captr.insptr _ &headptr;                           02763
                captr.fbstrptr _ (IF psptr.curnodeptr =
                psptr.begnodeptr THEN $"OK:" ELSE 0);              02764
                END;                                               02765
             # op:                                                 02766
                SIGNAL (interperr, $"Ambiguous Grammar --
                multiple $pca actions ");                          02767
          ENDCASE;                                                 02768
        = $dsel:                                                   02769
          BEGIN                                                    02770
          (dsellabel):                                             07157
          BUMP brcount;                                            02771
          CASE defaultptr.propcode OF                              02796
             = 0:    % not yet defined %                           02797
                BEGIN                                              02798
                defaultptr.propcode _ $getdae;                     02799
                defaultptr.insptr _ &headptr;                      02800
                defaultptr.fbstrptr _ (IF psptr.curnodeptr =
                psptr.begnodeptr THEN $"A:" ELSE 0);               02801
                END;                                               02802
             ENDCASE                                               02803
                SIGNAL (interperr, $"Ambiguous Grammar --
                multiple default actions ");                       02804
          IF nlmode = fulldisplay THEN                             02772
             BEGIN                                                 02774
             CASE captr.propcode OF                                02775
                = 0: % not yet defined %                           02776
                   BEGIN                                           02777
                   captr.propcode _ $getbug;                       02778
                   captr.insptr _ &headptr;                        02779
                   captr.fbstrptr _ (IF psptr.curnodeptr =
                   psptr.begnodeptr THEN $"B:" ELSE 0);            02780
                   END;                                            02781
                ENDCASE                                            02782
                   SIGNAL (interperr, $"Ambiguous Grammar --
                   multiple $pca actions ");                       02783
             END                                                   02793
          ELSE                                                     02794
             BEGIN                                                 02795
             CASE captr.propcode OF                                02805
                = 0: % not yet defined %                           02806
                   BEGIN                                           02807
                   captr.propcode _ $getdae;                       02808
                   captr.insptr _ &headptr;                        02809
                   captr.fbstrptr _ (IF psptr.curnodeptr =
                   psptr.begnodeptr THEN $"A:" ELSE 0);            02810
                   END;                                            02811
                ENDCASE                                            02812
                   SIGNAL (interperr, $"Ambiguous Grammar --
```

```
                    multiple ca actions ");                    02813
                END;                                           02814
            END;                                               02815
        = $ssel:                                               02816
            BEGIN                                              02817
            (ssellabel):                                       07158
            BUMP brcount;                                      02818
            CASE defaultptr.propcode OF                        02852
               = 0:    % not yet defined %                     02853
                  BEGIN                                        02854
                  defaultptr.propcode _ $getdae;               02855
                  defaultptr.insptr _ &headptr;                02856
                  defaultptr.fbstrptr _ (IF psptr.curnodeptr =
                  psptr.begnodeptr THEN $"A:" ELSE 0);         02857
                  END;                                         02858
               ENDCASE                                         02859
                  SIGNAL (interperr, $"Ambiguous Grammar --
                  multiple default actions ");                 02860
            CASE optptr.propcode OF                            02870
               = 0:    % not yet defined %                     02871
                  BEGIN                                        02872
                  optptr.propcode _ $getlit;                   02873
                  optptr.insptr _ &headptr;                    02874
                  optptr.fbstrptr _ (IF psptr.curnodeptr =
                  psptr.begnodeptr THEN $"[T]:" ELSE 0);       02875
                  END;                                         02876
               ENDCASE                                         02877
                  SIGNAL (interperr, $"Ambiguous Grammar --
                  multiple optional actions ");                02878
            IF nlmode = fulldisplay THEN                       02819
               BEGIN                                           02821
               CASE captr.propcode OF                          02831
                  = 0: % not yet defined %                     02832
                     BEGIN                                     02833
                     captr.propcode _ $getbug;                 02834
                     captr.insptr _ &headptr;                  02835
                     captr.fbstrptr _ (IF psptr.curnodeptr =
                     psptr.begnodeptr THEN $"B:" ELSE 0);      02836
                     END;                                      02837
                  ENDCASE                                      02838
                     SIGNAL (interperr, $"Ambiguous Grammar --
                     multiple $pca actions ");                 02839
               END                                             02849
            ELSE                                               02850
               BEGIN                                           02851
               CASE captr.propcode OF                          02861
                  = 0: % not yet defined %                     02862
                     BEGIN                                     02863
                     captr.propcode _ $getdae;                 02864
                     captr.insptr _ &headptr;                  02865
                     captr.fbstrptr _ (IF psptr.curnodeptr =
                     psptr.begnodeptr THEN $"A:" ELSE 0);      02866
                     END;                                      02867
                  ENDCASE                                      02868
                     SIGNAL (interperr, $"Ambiguous Grammar --
                     multiple $pca actions ");                 02869
```

```
            END;                                                    02879
         END;                                                       02880
      = $lsel:                                                      02881
         BEGIN                                                      02882
         (lsellabel):                                               07159
         BUMP brcount;                                              02884
         CASE defaultptr.propcode OF                                02885
            = 0:    % not yet defined %                             02886
               BEGIN                                                02887
               defaultptr.propcode _ $getlit;                       02888
               defaultptr.insptr _ &headptr;                        02889
               defaultptr.fbstrptr _ (IF psptr.curnodeptr =
               psptr.begnodeptr THEN $"T:" ELSE 0);                 02890
               END;                                                 02891
            ENDCASE                                                 02892
               SIGNAL (interperr, $"Ambiguous Grammar --
               multiple default actions ");                         02893
         CASE optptr.propcode OF                                    02903
            = 0:    % not yet defined %                             02904
               BEGIN                                                02905
               optptr.propcode _ $getdae;                           02906
               optptr.insptr _ &headptr;                            02907
               optptr.fbstrptr _ (IF psptr.curnodeptr =
               psptr.begnodeptr THEN $"[A]:" ELSE 0);               02908
               END;                                                 02909
            ENDCASE                                                 02910
               SIGNAL (interperr, $"Ambiguous Grammar --
               multiple optional actions ");                        02911
         IF nlmode = fulldisplay THEN                               04950
            BEGIN                                                   04951
            CASE captr.propcode OF                                  04952
               = 0: % not yet defined %                             04953
                  BEGIN                                             04954
                  captr.propcode _ $getbug;                         04955
                  captr.insptr _ &headptr;                          04956
                  captr.fbstrptr _ (IF psptr.curnodeptr =
                  psptr.begnodeptr THEN $"B:" ELSE 0);              04957
                  END;                                              04958
               ENDCASE                                              04959
                  SIGNAL (interperr, $"Ambiguous Grammar --
                  multiple $pca actions ");                         04960
            END                                                     04961
         ELSE                                                       04962
            BEGIN                                                   04963
            CASE captr.propcode OF                                  04964
               = 0: % not yet defined %                             04974
                  BEGIN                                             04975
                  captr.propcode _ $getlit;                         04976
                  captr.insptr _ &headptr;                          04977
                  captr.fbstrptr _ (IF psptr.curnodeptr =
                  psptr.begnodeptr THEN $"T:" ELSE 0);              04978
                  END;                                              04979
               ENDCASE                                              04971
                  SIGNAL (interperr, $"Ambiguous Grammar --
                  multiple $pca actions ");                         04972
            END;                                                    04973
```

```
   END;                                                    02912
= $xecute:                                                 02913
   BEGIN                                                   02914
   (xlabel):                                               07160
   % stack the instptr into the lptrstk %                  02915
      lptrstk[ lptrx ] _ &instptr;                         02916
      IF (lptrx _ lptrx+1) > $ptrssize                     02917
         THEN SIGNAL (interperr, $"lptrstk
         overflowed");                                     02918
   % set up new instptr and headptr values %               02919
      &headptr _ &instptr _ instptr.addr;                  02920
   REPEAT CASE;                                            02921
   END;                                                    02922
= $call:      % arbitrary execution function %             02923
   BEGIN                                                   03311
   (calllabel):                                            07161
   % assume that this is the first alternative and has
   already "failed" as there is no way to predict what
   this one does %                                         02924
   trivial _ FALSE;                                        03306
   IF returnval = $parallel THEN returnval _ $serial;
                                                           02925
   END;                                                    03310
= $pfcall:    % parsing function %                         02926
   BEGIN                                                   03309
   (pflabel):                                              07162
   IF returnval = $parallel THEN returnval _
   $parsefunction;                                         02927
   trivial _ FALSE;                                        03307
   END;                                                    03308
ENDCASE                                                    02928
   BEGIN                                                   02929
   IF instptr.opcode = $option OR instptr.opcode =
   $enter THEN                                             02930
      CASE &instptr _ instptr.nsuccessor OF                02931
         = 0:                                              02932
            IF lptrx > 0 THEN                              02933
               BEGIN                                       02934
               &instptr _ lptrstk[lptrx _ lptrx-1];
                                                           02935
               REPEAT CASE;                                02936
               END                                         02937
            ELSE SIGNAL (interperr, $"Meaningless
            grammar");                                     02938
         ENDCASE                                           02939
            BEGIN                                          02940
            IF resetheader := FALSE THEN                   02941
               &headptr _ &instptr;                        02942
            REPEAT CASE 2;                                 02943
            END;                                           02944
   % ***** have a "trivial alternative" *** (often a
   mistake in the grammar) ***** lets execute it and
   hope the writer of the cml program will fix up his
   code when he can't reach more meaningful
   alternatives %                                          02945
   IF trivial THEN                                         02951
```

```
                    BEGIN                                                02952
                    psptr.curnodeptr _ &instptr;                         02953
                    FOR i _ 0 UP UNTIL >= lptrx DO                       02954
                       BEGIN                                             02955
                       ptrstk[ptrx] _ lptrstk[i];                        02956
                       BUMP ptrx;                                        02957
                       END;                                              02958
                    RETURN( $serial, 1 );                                02959
                    END;                                                 02960
                 END;                                                    02961
              % get the next alternative (of the instptr) %              02962
                 CASE &headptr _ instptr.alternative OF                  02963
                    = 0:                                                 02964
                       IF lptrx > 0 THEN                                 02965
                          BEGIN                                          02966
                          &instptr _ lptrstk[lptrx _ lptrx-1];           02967
                          REPEAT CASE;                                   02968
                          END;                                           02969
                    = &instptr: % check for loop condition %             02970
                       BEGIN                                             02971
                       &headptr _ 0;                                     02972
                       REPEAT CASE (0);                                  02973
                       END;                                              02974
                 ENDCASE;                                                02975
              % set headptr to point to the head of the alternative
              chain %                                                    02976
                 &instptr _ &headptr;                                    02977
              END;                                                       02978
        RETURN (returnval, brcount);                                     02979
        END.
                                                                         02980
    (evaler) PROCEDURE( % sets up for and interprets an instruction %
                                                                         08934
        % FORMAL ARGUMENTS %                                             08935
           curptr);    % ptr to path stack entry %                        08936
        % NORMAL RETURNS %                                               08937
           % none %                                                       08938
        % ABNORMAL RETURNS %                                             08939
        LOCAL % VARIABLES %                                              08940
           count,                % count of arguments processed %         08941
           function,             % ptr to execution function %            08942
           instptr,              % ptr to the current instruction %       08943
           op,           % opcode of current inst %                      08944
           options,              % count of alternatives %                08945
           proc,                 % ptr to $pfcall parsing function %
                                                                         08946
           stateptr,             % ptr to the function state record %
                                                                         08947
           lptrx,        % index of next location in lptrstk %           08948
           % lptrstk SHOULD BE ptrssize (in PDATA) WORDS LONG %          08949
              lptrstk[40], % work stack for tree chasing %               08950
           tempptr,              % misc temp. ptr %                       08951
           arg[8];               % argument collection vector %           08952
        LOCAL TEXT POINTER tp1, tp2;                                     08953
        % REF VARIABLES %                                                08954
           REF fbstr, inpt, notetime;                                    08955
```

```
   REF tda, sysmsg, hlpcmdstk;                                    08956
   REF                                                            08957
      curptr,                                                     08958
      function,                                                   08959
      instptr,          % ptr to the current instruction %        08960
      proc,                                                       08961
      stateptr,                 % ptr to the function state
      record %                                                    08962
      tempptr;                                                    08963
LOCAL STRING                                                      08964
   helpstr[25], work[10];       % prompts string for $pfcall
   functions %                                                    08965
%----------------------%                                         08966
(again): % come here after optional construct is taken %         08967
% initialize local variables %                                   08968
   &instptr _ curptr.curnodeptr;                                  08969
   op _ instptr.opcode;                                           08970
% interrogate the alternatives to find out how to interpret the
next instruction if the next instruction has alternatives %
                                                                  08971
IF NOT cueflg AND ( instptr.alternative OR op IN
[$keyop,$xecute]) THEN                                            08972
   CASE curptr.evalmod _ sleuth( &curptr, $optaction,
   $caaction, $defaction :options ) OF                            08973
      = $parallel:     % recognizer may be invoked %              08974
         BEGIN                                                    08975
         % prompt the user if appropriate %                       08976
            % put together a feedback string for prompting %
                                                                  08977
               edistr(  $prompstr );                              08978
         % set up for parameter recognition %                     08979
            fbctl( startparams );                                 08980
         % look to see what is coming next if required %         08981
            IF options > 1 THEN                                   08982
               BEGIN                                              08983
               % output the feedback string %                    08984
                  IF inprompts # noprompts AND NOT cueflg THEN
                                                                  08985
                     fbctl( incues, $prompstr );                  08986
               cueflg _ TRUE;                                     08987
               CASE lookc() OF                                    08988
                  = rptchar:                                      08989
                     BEGIN                                        08990
                     IF msjfn THEN  %monitoring commands%         08991
                        msrecord(mrepeat, 0, 0);                  08992
                     IF pathx > $totalrecsize                     08993
                        THEN GOTO cacharcase;                     08994
                     inpt(); % read the char %                    08995
                     % return last cmd chars to input buffer %
                                                                  08996
                        reseth( $keyrptstr );                     08997
                     REPEAT CASE;                                 08998
                     END;                                         08999
                  = inschar:                                      09000
                     BEGIN                                        09001
                     IF msjfn THEN  %monitoring commands%         09002
```

```
                msrecord(minsert, 0, 0);                    09003
             IF pathx > $totalrecsize                       09004
                THEN GOTO cacharcase;                       09005
             % check to make sure that the current
             system is the nlseditor, %                     09006
                IF [$sbstack + sbstkx -
                $sbentsize].sbptr # $nlseditor              09007
                   THEN GOTO cacharcase;                    09008
             inpt(); % read the char %                      09009
             &instptr _ $zinsstatement;                     09010
             cueflg _ FALSE;                                09011
             END;                                           09012
          = cachar:                                         09013
             (cacharcase):                                  09014
             IF (&instptr _ caaction.insptr) = 0            09015
                THEN                                        09016
                   BEGIN                                    09017
                   inpt();                                  09018
                   fbctl( '? );                             09019
                   REPEAT CASE;                             09020
                   END;                                     09021
          = optchar:                                        09022
             IF (&instptr _ optaction.insptr) = 0           09023
                THEN                                        09024
                   BEGIN                                    09025
                   inpt();                                  09026
                   fbctl( '? );                             09027
                   REPEAT CASE;                             09028
                   END                                      09029
                ELSE                                        09030
                   BEGIN                                    09031
                   cueflg _ FALSE;                          09032
                   IF optaction.propcode IN
                   [$option,$anyof]                         09033
                      THEN                                  09034
                         BEGIN                              09035
                         inpt();                            09036
                         % romp down the current path
                         stacking ptrs to execute
                         instructions if any %              09037
                         &tempptr _ curptr.curnodeptr;
                                                            09038
                         WHILE tempptr.opcode = $xecute
                         DO                                 09039
                         BEGIN                              09040
                         % stack ptr to execute inst
                         into ptrstk %                      09041
                         ptrstk[ ptrx ] _ &tempptr;
                                                            09042
                         IF (ptrx _ ptrx + 1) >
                         $ptrssize                          09043
                            THEN SIGNAL (interperr,
                            $"ptrstk overflowed");
                                                            09044
                         &tempptr _ tempptr.addr;           09045
                         END;                               09046
```

```
                                % stack ptr to optional inst
                                into ptrstk %                    09047
                                   ptrstk[ ptrx ] _ &instptr;
                                                                 09048
                                   IF (ptrx _ ptrx + 1) >
                                   $ptrssize                     09049
                                   THEN SIGNAL (interperr,
                                   $"ptrstk overflowed");
                                                                 09050
                                % evaluate the optional inst.
                                %                                09051
                                   curptr.curnodeptr _
                                   instptr.addr;                 09052
                                GOTO again;                      09053
                                END;                             09054
                             END;                                09055
                       = BC, =BW:                                09056
                          BEGIN                                  09057
                          % Catch BC and BW here ?????????? %    09058
                          inpt();                                09059
                          IF cutstop THEN                        09060
                             BEGIN                               09061
                             fbctl('?);                          09062
                             REPEAT CASE;                        09063
                             END;                                09064
                          SIGNAL (popstate);                     09065
                          END;                                   09066
                       = CD:                                     09067
                          BEGIN                                  09068
                          inpt();                                09069
                          SIGNAL (cmdelete);                     09070
                          END;                                   09071
                       = '?:                                     09072
                          BEGIN                                  09073
                          inpt();                                09074
                          &fbstr _ $"";                          09075
                          fbhelp( &curptr, cmdmode );            09076
                          REPEAT CASE;                           09077
                          END;                                   09078
                       = 'S-100B:  % <^S> %                      09079
                          BEGIN                                  09080
                          inpt();                                09081
                          &fbstr _ $"";                          09082
                          cshelp( &curptr, cmdmode, TRUE );      09083
                          REPEAT CASE;                           09084
                          END;                                   09085
                    ENDCASE                                      09086
                       IF (&instptr _ defaction.insptr) = 0      09087
                          THEN                                   09088
                             BEGIN                               09089
                             inpt();                             09090
                             fbctl( '? );                        09091
                             REPEAT CASE;                        09092
                             END;                                09093
                    END;                                         09094
              % update the curnodeptr field in the path stack record
```

```
        %                                                       09095
           curptr.curnodeptr _ &instptr;                        09096
           op _ instptr.opcode;                                 09097
        END;                                                    09098
      = $serial:       % execution function %                   09099
        NULL;                                                   09100
      = $parsefunction:        % parsing function %            09101
        BEGIN                                                   09102
        % romp down the current path stacking ptrs to execute
        instructions if any %                                   09103
           &instptr _ curptr.curnodeptr;                        09104
           WHILE instptr.opcode = $xecute DO                    09105
              BEGIN                                             09106
              % stack ptr to execute inst into ptrstk %         09107
              ptrstk[ ptrx ] _ &instptr;                        09108
              IF (ptrx _ ptrx + 1) > $ptrssize                  09109
                 THEN SIGNAL (interperr, $"ptrstk
                 overflowed");                                  09110
              &instptr _ instptr.addr;                          09111
              END;                                              09112
           curptr.curnodeptr _ &instptr;                        09113
        % build prompt string for builtin functions %           09114
           % put together a feedback string for prompting %
                                                                09115
           edistr( $prompstr );                                 09116
        % call parse functions in "parsehelp" mode to solicit
        a subhelp string %                                      09117
           lptrx _ 0;                                           09118
           *helpstr* _ NULL;                                    09119
           WHILE &instptr DO                                    09120
              BEGIN                                             09121
              CASE op _ instptr.opcode OF                       09122
                 = $xecute:                                     09123
                    BEGIN                                       09124
                    % stack the instptr into the lptrstk %
                                                                09125
                       lptrstk[ lptrx ] _ &instptr;             09126
                       IF (lptrx _ lptrx+1) > $ptrssize         09127
                          THEN SIGNAL (interperr, $"lptrstk
                          overflowed");                         09128
                    % set up new instptr values %              09129
                       &instptr _ instptr.addr;                 09130
                    REPEAT CASE;                                09131
                    END;                                        09132
                 = $pfcall:    % parsing function %             09133
                    BEGIN                                       09134
                    &proc _ instptr.addr;                       09135
                    *work* _ NULL;                              09136
                    proc( &curptr, parsehelp, $work );          09137
                    IF work.L THEN                              09138
                       IF helpstr.L THEN                        09139
                          *helpstr* _ *helpstr*, "/, *work*
                                                                09140
                       ELSE                                     09141
                          *helpstr* _ *work*;                   09142
                    END;                                        09143
```

```
                = $option :                                          09144
                   CASE &instptr _ instptr.nsuccessor OF
                                                                     09145
                      = 0:                                           09146
                         IF lptrx > 0 THEN                           09147
                            BEGIN                                    09148
                            &instptr _ lptrstk[lptrx _
                            lptrx-1];                                09149
                            REPEAT CASE;                             09150
                            END                                      09151
                         ELSE SIGNAL (interperr,
                         $"Meaningless grammar");                    09152
                      ENDCASE                                        09153
                         BEGIN                                       09154
                         REPEAT CASE 2;                              09155
                         END;                                        09156
                   ENDCASE;                                          09157
             % get the next alternative (of the instptr) %
                                                                     09158
                   CASE instptr.alternative OF                       09159
                      = 0:                                           09160
                         IF lptrx > 0 THEN                           09161
                            BEGIN                                    09162
                            &instptr _ lptrstk[lptrx _ lptrx-1];
                                                                     09163
                            REPEAT CASE;                             09164
                            END                                      09165
                         ELSE &instptr _ 0;                          09166
                      = &instptr: % check for loop condition %
                                                                     09167
                         BEGIN                                       09168
                         REPEAT CASE (0);                            09169
                         END;                                        09170
                      ENDCASE &instptr _ instptr.alternative;
                                                                     09171
                END;                                                 09172
             &instptr _ curptr.curnodeptr;                           09173
          % append helpstr in front of the prompts string %
                                                                     09174
             IF helpstr.L > 0 THEN                                   09175
                BEGIN                                                09176
                LOOP                                                 09177
                   BEGIN                                             09178
                   IF NOT                                            09179
                      FIND SF(*helpstr*) [":] ^tp1 ^tp2 _tp1
                      THEN EXIT LOOP;                                09180
                   *helpstr* _                                       09181
                      SF(*helpstr*) tp1, tp2 SE(*helpstr*);
                                                                     09182
                   END;                                              09183
                IF prompstr.L > 0 THEN                               09184
                   *prompstr* _ *helpstr*, "/, *prompstr*            09185
                ELSE *prompstr* _ *helpstr*, ":;                     09186
                END;                                                 09187
          % output the prompt (after checking for "OK/B") %
                                                                     09188
```

```
            IF FIND SF(*prompstr*) ["CK/B"] ^tp2 < 4CH > ^tp1
            THEN                                                    09841
               ST tp1 tp2 _ "OK";                                   09842
            fbctl( incues, $prompstr );                             09189
            cueflg _ TRUE;                                          09190
         % check next inpt char for trigger action %                09191
            CASE lookc() OF                                         09192
               = '?:                                                09193
                  BEGIN                                             09194
                  inpt();                                           09195
                  &fbstr _ $"";                                     09196
                  fbhelp( &curptr, cmdmode );                       09197
                  REPEAT CASE;                                      09198
                  END;                                              09199
               = 'S-100B:  % <^S> %                                 09200
                  BEGIN                                             09201
                  inpt();                                           09202
                  &fbstr _ $"";                                     09203
                  cshelp( &curptr, cmdmode, TRUE );                 09204
                  REPEAT CASE;                                      09205
                  END;                                              09206
               = BC, = BW:                                          09207
                  BEGIN                                             09208
                  inpt();                                           09209
                  IF cutstop THEN                                   09210
                     BEGIN                                          09211
                     fbctl('?);                                     09212
                     REPEAT CASE;                                   09213
                     END;                                           09214
                  SIGNAL( popstate );                               09215
                  END;                                              09216
               = CD:                                                09217
                  BEGIN                                             09218
                  inpt();                                           09219
                  SIGNAL( cmdelete );                               09220
                  END;                                              09221
               ENDCASE;                                             09222
         END;                                                       09223
      ENDCASE SIGNAL (interperr, $"Unexpected evalmode value");
                                                                    09224
   % decode the current node in the grammar %                       09225
      % the decoding process sets up the path stack record
      (curptr) for evaluation by the functional execution process.
      Some of the "builtin" processes are transformed into
      function execution processes for calls on external routines
      %                                                             09226
      decode(&curptr);                                              09227
   % process the current node %                                     09228
      % if the pfunction field of the path stack record is not
      NULL, then the arguments are collected and the function is
      invoked %                                                     09229
      IF (&function _ curptr.pfunction) # 0                         09230
         THEN                                                       09231
            BEGIN                                                   09232
            % collect ptrs to arguments to arg vector %            09233
               FOR count _ curptr.argcount DOWN UNTIL < 1 DO        09234
```

```
                arg[count] _ xpop();                                    09235
            % set the stateptr to point to the function state
            record %                                                    09236
               &stateptr _ &curptr + $pathrecsize;                      09237
            % $call processing routine, pushing ptr to current
            node onto eval stack if the routine returns TRUE %
                                                                        09238
            IF analyzing AND (op = $call) AND (curptr.pmode =
            parsing)  THEN                                              09239
               BEGIN                                                    09240
               !time();                                                 09241
               notetime(102B %command execute%, r1);                    09242
               END;                                                     09243
            (xrcall):                                                   09244
            IF function(&stateptr, curptr.pmode, arg[1], arg[2],
            arg[3], arg[4], arg[5], arg[6], arg[7], arg[8])  09245
               THEN                                                     09246
                  BEGIN                                                 09247
                  curptr.pmode _ pnext;                                 09248
                  IF op NOT IN [$fbclear, $store] THEN                  09249
                     xpush( &stateptr );                                09250
                  % clear cueflg if the instruction just executed
                  was a recognizer %                                    09251
                     IF op IN [$keyop,$call]                            09252
                        THEN cueflg _ FALSE;                            09253
                  END                                                   09254
               ELSE                                                     09255
                  curptr.pmode _ backup;                                09256
            END                                                         09257
         ELSE curptr.pmode _ pnext;                                     09258
   RETURN;                                                              09259
   END.

                                                                        09260
(testselect) PROCEDURE(                                                 01522
   % FUNCTION %                                                         01523
      % this routine checks to see if the path stack entry
      indicated by curptr points to a selection function which can
      be backed up. %                                                   01524
   % FORMAL ARGUMENTS %                                                01525
      curptr);    % ptr to path stack entry %                         01526
   % RETURNS %                                                         01527
      % 1) TRUE/FALSE  boolean condition %                            01528
   LOCAL % VARIABLES %                                                  01529
      statesavptr;        % ptr to selection state info %             01530
   REF % VARIABLES %                                                    01531
      statesavptr,        % ptr to selection state info %             01532
      curptr;     % ptr to path stack entry %                         01533
   %-------------%                                                     01534
   IF [curptr.curnodeptr].opcode IN [$ssel, $lsel] THEN                 01535
      BEGIN                                                             01536
      &statesavptr _ &curptr + $pathrecsize + psellen;                  01537
      IF statesavptr.nselects > 0 THEN                                  01538
         BEGIN                                                          01539
         curptr.pmode _ popselect;                                      01540
         ptrx _ curptr.ptrxsav;                                         01541
         evalx _ curptr.evalxsav - 1;                                   01542
```

```
            RETURN (TRUE);                                          01543
            END;                                                    01544
         END;                                                       01545
      RETURN (FALSE);                                               01546
      END.
                                                                    01547
   (decode) PROCEDURE(                                              01548
      % FUNCTION %                                                  01549
         % decode decodes the interpretive instruction indicated by
         the curnodeptr field of the path stack entry indicated by
         psptr and either executes the instruction if it is simple
         enough or sets up the path stack entry for subsequent
         function invocation %                                      01550
         % the opcodes processed locally require no saving of state
         information and no processing during backup operations and
         include the following list: ENTER, LOAD, STORE, VALUEOF%
                                                                    01551
      % FORMAL ARGUMENTS %                                         01552
         psptr);     % ptr to path stack entry %                  01553
      % NORMAL RETURNS %                                           01554
         % none %                                                  01555
      % ABNORMAL RETURNS %                                        01556
         % SIGNALS ARE GENERATED AS FOLLOWS: %                    01557
            % 1) interperr -- interpreter error %                 01558
      LOCAL % VARIABLES %                                         01559
         args,       % number of arguments %                     01560
         functaddr,  % address of the processing function %      01561
         funstateptr,        % ptr to function state record %    01562
         instptr,    % pointer to current instruction %          01563
         lastptr,    % ptr to previous path stack entry %        01564
         op, % interpreter function code %                       01565
         temp,       % temporary value %                         01566
         tempptr,    % scratch pointer %                         01567
         val;        % value/address field of interpreter word %
                                                                    01568
      REF % VARIABLES %                                           01569
         funstateptr,                                             01570
         instptr,                                                 01571
         lastptr,                                                 01572
         psptr,                                                   01573
         tempptr;                                                 01574
      %----------------------%                                    01575
      % initialize local variables %                             01576
         args _ functaddr _ 0;                                    01577
         &funstateptr _ &psptr + $pathrecsize;                    01578
      (over):                                                     01579
      % strip apart the instruction %                            01580
         &instptr _ psptr.curnodeptr;                             01581
         op _ instptr.opcode;                                     01582
         val _ instptr.addr;                                      01583
      % process by function code %                               01584
         CASE op OF                                               01585
            % RECOGNIZERS %                                       01586
               = $keyop:     % keyword recognition operator %     01587
                  BEGIN                                           01588
                  % construct a single argument whose value is the
```

```
        pointer to the path stack entry %                          01589
           xpush( &psptr );                                        01590
           args _ 1;                                               01591
           functaddr _ $keywrec;                                   01592
        END;                                                       01593
     = $dsel,      % destination selection %                       01594
     = $lsel,      % literal selection %                           01595
     = $ssel:      % source selection %                            01596
        BEGIN                                                      01597
        args _ 2;                                                  01598
        functaddr _ $xselect;                                      01599
        xpush( &psptr );                                           01600
        % set up maxselects and nselects:%                        01601
           &tempptr _ &funstateptr + 4;                            01602
           tempptr.maxselects _ tempptr.nselects _ 0;              01603
        END;                                                       01604
     = $vwspecs:   % gets viewspecs %                              01605
        BEGIN                                                      01606
        needconfirm _ FALSE;                                       01607
        functaddr _ $xviewspecs;                                   01608
        END;                                                       01609
     = $levadj:    % get level adjust string %                    01610
        BEGIN                                                      01611
        needconfirm _ FALSE;                                       01612
        functaddr _ $xlevadj;                                      01613
        END;                                                       01614
     = $confirm:   % get command confirmation %                    01615
        BEGIN                                                      01616
        functaddr _ $xconfirm;                                     01617
        % argument is opcode of previous instruction %
                                                                   01618
           xpush( lastsel );                                       01619
           args _ 1;                                               01620
        END;                                                       01621
     = $anyof:                                                     01622
        BEGIN                                                      01623
        % look to see if $anyof node is at the top of the
        eval stack, if not, then we must initialize an
        entry at the top of the stack %                            01624
           &lastptr _ xread() - $pathrecsize;                      01625
           IF [lastptr.curnodeptr].opcode # $anyof THEN
                                                                   01626
              BEGIN                                                01627
              xpush( &funstateptr );                               01628
              funstateptr _ 0;                                     01629
              END;                                                 01630
        END;                                                       01631
     = $option:                                                    01632
        NULL;                                                      01633
  % CONTROL ELEMENTS %                                            01634
     = $pfcall:    % parse function $call %                       01635
        BEGIN                                                      01636
        functaddr _ val;                                           01637
        args _ instptr.val2;                                       01638
        END;                                                       01639
     = $xecute:    % transfer to another point in tree %
```

```
                                                                    01640
            BEGIN                                                   01641
            IF defaction.propcode =  $keyop                         01642
               THEN REPEAT CASE ($keyop);                           01643
            % stack ptr to $xecute inst in ptrstk and evaluate
            the descendent path %                                   01644
               ptrstk[ ptrx ] _ &instptr;                           01645
               IF (ptrx _ ptrx + 1) > $ptrssize                     01646
                  THEN SIGNAL (interperr, $"ptrstk
                  overflowed");                                     01647
               psptr.curnodeptr _ instptr.addr;                     01648
               GOTO over;                                           01649
            END;                                                    01650
         = $call:      % subroutine $call %                       01651
            BEGIN                                                   01652
            % provide running feedback if in DNLS and user
            wants feedback %                                        01653
               IF nlmode = fulldisplay                              01654
                  AND inprompts # noprompts THEN                    01655
                     fbctl( incues, $"...");                        01656
            functaddr _ val;                                        01657
            args _ instptr.val2;                                    01658
            END;                                                    01659
      % FEEDBACK ELEMENTS %                                        01660
         = $fbclear:   % clear feedback buffer %                    01661
            BEGIN                                                   01662
            % set address to point to interface routine %          01663
               functaddr _ $xfeedback;                              01664
            % push arguments onto eval stack %                     01665
               xpush( $fbclear );                                   01666
               args _ 1;                                            01667
            END;                                                    01668
         = $necho:     % echo noise word string %                  01669
            BEGIN                                                   01670
            % set address to point to interface routine %          01671
               functaddr _ $xfeedback;                              01672
            % push arguments onto eval stack %                     01673
               xpush( $necho );                                     01674
               xpush( val );                                        01675
               args _ 2;                                            01676
            END;                                                    01677
         = $recho:     % replace last thing echoed %               01678
            BEGIN                                                   01679
            % set address to point to interface routine %          01680
               functaddr _ $xfeedback;                              01681
            % push arguments onto eval stack %                     01682
               xpush( $recho );                                     01683
               xpush( val );                                        01684
               args _ 2;                                            01685
            END;                                                    01686
      % VALUE MANIPULATIONS %                                      01687
         = $pload:     % $pload variable to ptr stack %            01688
            BEGIN                                                   01689
            temp _ [val]; % fetch contents of variable %           01690
            % check to make sure that the frame pointed to by
            the rh of temp still has a fcounter value which =
```

```
                the left half of temp %                                 01691
                   IF temp.LH # [temp.RH - $pathrecsize].fcounter
                                                                        01692
                      THEN SIGNAL ( interperr, $"reference to
                      undefined interpreter variable");                 01693
                % push the ptr to the value record onto the eval
                stack %                                                 01694
                   xpush( temp.RH );                                    01695
                END;                                                    01696
             = $store:     % $store value into variable %               01697
                BEGIN                                                   01698
                functaddr _ $xstor;                                     01699
                args _ 1;                                               01700
                xpush( val );                                           01701
                END;                                                    01702
             = $enter:     % $enter constant into stack %               01703
                BEGIN                                                   01704
                xpush( &funstateptr );                                  01705
                funstateptr.a1word _ (IF instptr.val2 THEN
                instptr.val2 ELSE val);                                 01706
                funstateptr.a2word _ val;                               07746
                END;                                                    01707
          ENDCASE SIGNAL(interperr, $"unrecognizable interpreter op
          code");                                                       01708
       % terminate processing, return to caller %                       01709
          psptr.pfunction _ functaddr;                                  01710
          psptr.argcount _ args;                                        01711
          RETURN;                                                       01712
       END.
                                                                        01713
% EVAL STACK MANIPULATION ROUTINES %                                    01714
   (xpush) PROCEDURE(                                                   01715
      % pushes the address of an argument record onto eval stack %
                                                                        01716
      % FORMAL ARGUMENTS %                                             01717
         addrofvalue ); % address of value %                            01718
      % NORMAL RETURNS %                                               01719
         % none %                                                       01720
      % ABNORMAL RETURNS %                                             01721
         % SIGNAL interperr generated for eval stack overflow %        01722
      %---------------------%                                          01723
      IF evalx NOT IN [0,$evalsize)                                     01724
         THEN SIGNAL (interperr, $"Eval stack out of range");           01725
      evalstk[ evalx ] _ addrofvalue;                                   01726
      BUMP evalx;                                                       01727
      RETURN;                                                           01728
      END.
                                                                        01729
   (xpop) PROCEDURE;                                                    01730
      % pops the address of an argument record from the eval stack %
                                                                        01731
      % FORMAL ARGUMENTS %                                             01732
         % none %                                                       01733
      % NORMAL RETURNS %                                               01734
         % address of argument record contained in the top of the
         eval stack %                                                   01735
```

```
      % ABNORMAL RETURNS %                                              01736
         % SIGNAL interperr generated for eval stack underflow %
                                                                        01737
      %---------------------%                                          01738
      IF evalx NOT IN (0,$evalsize]                                     01739
         THEN SIGNAL (interperr, $"Eval stack out of range");           01740
      evalx _ evalx - 1;                                                01741
      RETURN (evalstk[evalx]);                                          01742
      END.
                                                                        01743
   (xstore) PROCEDURE(                                                  01744
      % stores the address of an argument record onto eval stack %
                                                                        01745
      % FORMAL ARGUMENTS %                                             01746
         addrofvalue ); % address of value %                            01747
      % NORMAL RETURNS %                                               01748
         % none %                                                       01749
      % ABNORMAL RETURNS %                                             01750
         % SIGNAL interperr generated for eval stack out of range %
                                                                        01751
      %---------------------%                                          01752
      IF evalx NOT IN (0,$evalsize]                                     01753
         THEN SIGNAL (interperr, $"Eval stack out of range");           01754
      evalstk[evalx-1] _ addrofvalue;                                   01755
      RETURN;                                                           01756
      END.
                                                                        01757
   (xread) PROCEDURE;                                                   01758
      % reads the address of an argument record from the eval stack %
                                                                        01759
      % FORMAL ARGUMENTS %                                             01760
         % none %                                                       01761
      % NORMAL RETURNS %                                               01762
         % address of argument record contained in the top of the
         eval stack %                                                   01763
      % ABNORMAL RETURNS %                                             01764
         % SIGNAL interperr generated for eval stack out of range %
                                                                        01765
      %---------------------%                                          01766
      IF evalx NOT IN (0,$evalsize]                                     01767
         THEN SIGNAL (interperr, $"Eval stack out of range");           01768
      RETURN (evalstk[evalx-1]);                                        01769
      END.
                                                                        01770
% I/O SUPPORT ROUTINES %                                               01771
   (edistr) PROCEDURE( % edits terminal prompt string %                 01772
      % FORMAL ARGUMENTS %                                             01773
         targetptr);% ptr to result string %                            01774
      % RETURNS %                                                      01775
         % none %                                                       01776
      % ABNORMAL RETURNS %                                             01777
         % none %                                                       01778
      LOCAL % VARIABLES %                                               01779
         i,          % loop counter %                                  01780
         sourceptr;  % ptr to source string %                           01781
      LOCAL TEXT POINTER tp1, tp2;                                      04926
```

```
      REF % VARIABLES %                                                  01782
         sourceptr,  % ptr to argument string %                          01783
         targetptr;  % ptr to result string %                            01784
      %-----------------%                                                01785
      *targetptr* _ NULL;                                                01786
      % set sourceptr to alternatively point to each of the three
      possible prompt strings, and add these strings to the target
      string if appropriate %                                            01787
         FOR i _ 1 UP UNTIL > 3 DO                                       01788
            BEGIN                                                        01789
            &sourceptr _ CASE i OF                                       01790
               = 1: caaction.fbstrptr;                                   01792
               = 2: defaction.fbstrptr;                                  01791
               = 3: optaction.fbstrptr;                                  01793
               ENDCASE 0;                                                01794
            IF &sourceptr # 0 THEN                                       01795
               BEGIN                                                     01796
               % exit if we are ignoring optional prompts and the
               prompt string contains a $option character (*) %          01797
                  IF inprompts = partprompts                             01798
                     AND (FIND SF(*sourceptr*) C'C])                     01799
                        THEN REPEAT LOOP;                                01800
               CASE targetptr.L OF                                       01801
                  > 0:                                                   01802
                     IF NOT FIND SF(*targetptr*) C*sourceptr*]           01803
                        THEN *targetptr* _ *targetptr*, "/",
                        *sourceptr*                                      01804
                  = 0:                                                   01805
                     *targetptr* _ *sourceptr*;                          01806
               ENDCASE;                                                  01807
               LOOP                                                      04927
                  BEGIN                                                  04928
                  IF NOT                                                 04929
                     FIND SF(*targetptr*) C':] ^tp1 ^tp2 _tp1            04930
                     THEN EXIT LOOP;                                     04931
                  *targetptr* _ SF(*targetptr*) tp1, tp2
                  SE(*targetptr*);                                       04932
                  END;                                                   04933
               *targetptr* _ *targetptr*, ':;                            04934
               END;                                                      01808
            END;                                                         01809
      RETURN;                                                            01810
      END.

                                                                         01811
% BUILTIN INTERPRETER FUNCTIONS %                                        01812
   (xfeedback) PROCEDURE(                                                01813
      % provide feedback elements for CML %                              01814
      % FORMAL ARGUMENTS %                                               01815
         resultptr,  % ptr to the result record %                        01816
         parsemode,  % command parsing mode %                            01817
         fbmode,     % feedback mode %                                    01818
         fbstring ); % pointer to feedback string %                      01819
      % NORMAL RETURNS %                                                 01820
         % 1) returns the result ptr %                                   01821
      % ABNORMAL RETURNS %                                               01822
         % generates a SIGNAL if the feedback mode is not recognized
```

```
      %                                                          01823
   REF % VARIABLES %                                             01824
      resultptr,                                                 01825
      fbstring;                                                  01826
   %-----------------%                                          01827
   CASE parsemode OF                                             01828
      = parsing:                                                 01829
         BEGIN                                                   01830
         % save the current length of the feedback buffer in the
         function state record (pointed to by resultptr) %       01831
            resultptr[1] _ cflstr.L;                             01832
         CASE fbmode OF                                          01833
            = $necho:                                            01834
               BEGIN                                             01835
               fbctl( echostr, &fbstring );                      01836
               END;                                              01837
            = $recho:                                            01838
               BEGIN                                             01839
               fbctl( rechostr, &fbstring );                     01840
               END;                                              01841
            = $fbclear:                                          01842
               BEGIN                                             01843
               fbctl( clearcfl );                                01844
               END;                                              01845
            ENDCASE SIGNAL (interperr, $"unrecognized feedback
            code in xfeedback");                                 01846
         cueflg _ FALSE;                                         07749
         END;                                                    01847
      = cleanup,                                                 01848
      = backup:                                                  01849
         BEGIN                                                   01850
         % restore the command feedback line length %            01851
            cflstr.L _ resultptr[1];                             01852
         END;                                                    01853
      ENDCASE;                                                   01854
   RETURN (&resultptr);                                          01855
   END.
                                                                 01856
(xstor) PROCEDURE(                                               01857
   % perform the $store operation %                              01858
   % FORMAL ARGUMENTS %                                         01859
      resultptr,  % ptr to the result record %                   01860
      parsemode,  % command parsing mode %                       01861
      location);  % ptr to $store location %                     01862
   % NORMAL RETURNS %                                           01863
      % 1) returns the result ptr %                              01864
   LOCAL % VARIABLES %                                           01865
      frameptr;   % ptr to current frame %                       01866
   REF % VARIABLES %                                             01867
      frameptr,                                                  01868
      resultptr,                                                 01869
      location;                                                  01870
   %-----------------%                                          01871
   CASE parsemode OF                                             01872
      = parsing:                                                 01873
         BEGIN                                                   01874
```

```
            % fetch the record pointer contained in the top of the
            eval stack, save this away in the function state record
            (for backup purposes) %                                  01875
               resultptr _ xpop();                                   01876
            % set frameptr to point to the path stack record
            associated with that frame %                             01877
               &frameptr _ resultptr - $pathrecsize;                 01878
            % pack the frame counter for the frame with the address
            of the result record %                                   01879
               resultptr.LH _ frameptr.fcounter;                     01880
            % save the augmented ptr away in the variable %         01881
               location _ resultptr;                                 01882
            END;                                                     01883
         = cleanup,                                                  01884
         = backup:                                                   01885
            BEGIN                                                    01886
            % push the ptr to the saved frame back onto the eval
            stack %                                                  01887
               % set ptr to frame %                                  01888
                  &frameptr _ &resultptr - $pathrecsize;             01889
               % set evalx so eval stack will look like it did before
               the $store operation was executed %                   01890
                  evalx _ frameptr.evalxsav - 1;                     01891
               % push ptr to result record back onto eval stack %
                                                                     01892
                  xpush( resultptr.RH );                             01893
            END;                                                     01894
      ENDCASE;                                                       01895
      RETURN (&resultptr);                                           01896
      END.

                                                                     01897
% KEYWORD RECOGNITION AND FEEDBACK SUPPORT %                         01898
   % KEYWORD RECOGNITION ROUTINE %                                   01899
      (keywrec) PROC(                                                09261
         % FORMAL ARGUMENTS %                                        09262
            resultptr,  % ptr to result record %                     09263
            parsemode,  % parsing mode %                             09264
            curptr );   % ptr to path stack entry %                  09265
         % FUNCTION %                                                09266
            % this routine performs all recognition and feedback
            functions for identifying a command keyword.  Four
            recognition methods are presently supported:
                1) recognition of some subset of the keyword list by an
            initial character
                2) recognition of all elements of the keyword list by a
            minimim unique initial substring.                        09267
                3) mdemand recognition requiring a right delimiter.
                                                                     09268
                4) recognition based upon a fixed number of characters.
                                                                     09269
            The second recognition mode is entered if either an initial
            left delimiter (SP) is encountered.  A question mark may be
            typed at any point, and the keywords still available for
            recognition are printed out for the user. %              09270
         % NORMAL RETURNS %                                          09271
            % this routine returns the vocabulary index of the
```

```
      recognized keyword.  If recognition fails, a value of 0 is
      returned.  If the nofail $option is selected (nofail =
      TRUE), then control remains in this routine until some valid
      keyword is recognized %                                    09272
   % ABNORMAL RETURNS %                                          09273
      % recognition of CD causes a GOTO STATE %                  09274
      % typing BC or BW when the feedback buffer is empty is
      interpretted as a request to back up the parse, and a signal
      named "popstate" is generated %                            09275
   LOCAL                                                         09276
      list,          % ptr to head of keyword list %             09277
      recogflag,     % TRUE if have recognized keyword %         09278
      fullflag,      % TRUE if alternate recognition is emplyed %
                                                                 09279
      echotype,      % type of echoing after recognition %       09280
      cmode,         % local copy of cmdmode %                   09281
      recmode,       % working recog. mode (recogmode/recog2mode) %
                                                                 09282
      nextchar,      % next inpt character %                     09283
      hits,          % number of compares %                      09284
      hindex,        % hit index into vocab of string %          09285
      hitx,          % temp hit index %                          09286
      oldptrx,       % starting value for ptrx %                 09287
      % work SHOULD BE wasize (in PDATA) WORDS LONG %           09288
         work[30];          % state buffer for keyinit and nextkey %
                                                                 09289
   REF                                                           09290
      list,                                                      09291
      curptr,                                                    09292
      resultptr;                                                 09293
   REF fbstr, inpt;                                              09294
   REF tda, sysmsg, hlpcmdstk;                                   09295
   %---------------------%                                       09296
   kwrstate _ FALSE;                                             09297
   CASE parsemode OF                                             09298
      = parsing:                                                 09299
         BEGIN                                                   09300
         % set up some state info (change later) %               09301
            &list _ curptr.curnodeptr;                           09302
            oldptrx _ ptrx;                                      09303
            recogflag _ fullflag _  FALSE;                       09304
            recmode _ recogmode;                                 09305
            echotype _ 1; % full echoing of keyword %            09306
            cmdmode _ 0;                                         09307
            IF nlmode                                            09308
               THEN cmdmode.dnlscmd _ TRUE                       09309
               ELSE cmdmode.tnlscmd _ TRUE;                      09310
            IF recmode = mexpert                                 09311
               THEN cmdmode.llcmd _ TRUE;                        09312
            cmode _ cmdmode;                                     09313
            % set up fbstr in 5th and subsequent words of work
            area %                                               09314
               &fbstr _ $work + 4;                               09315
               fbstr.M _ $fbstrmax; fbstr.L _ 0;                 09316
            % save away current state values %                  09317
               resultptr.keyinlength _ keyinpstr.L; % save current
```

```
            length of keyword save string %                    09318
            resultptr.ksaveflag _ keysaveflag;                 09319
            resultptr.fblen _ cflstr.L;                        09320
         % perfom any global functions associated with keyword
         recognition %                                         09321
            fbctl( startrec );                                 09322
         % prompt the user if not already done so %            09323
            IF NOT cueflg THEN fbctl( incues, $"C:" );         09324
      % read next character and look for control characters %
                                                               09325
         CASE nextchar _ inpcuc() OF                           09326
            =SP:       % left/right delimiter character %      09327
               BEGIN                                           09328
               kwrstate _ 1;                                   09329
               CASE recmode OF                                 09330
                  = mexpert:                                   09331
                     % check for use as a left delimiter %
                                                               09332
                        IF fbstr.L = 0 AND NOT fullflag        09333
                           THEN                                09334
                              BEGIN %  start of alternate
                              recognition %                    09335
                              IF nlmode NOT= fulldisplay THEN
                              typech(SP);                      09336
                              fullflag _ TRUE;                 09337
                              cmode.llcmd _ FALSE;             09338
                              recmode _ recog2mode;            09339
                              REPEAT CASE 2;                   09340
                              END;                             09341
                  = mdemand:                                   09342
                     % check for a right delimiter %           09343
                        IF recogflag                           09344
                           THEN                                09345
                              BEGIN                            09346
                              echotype _ 2; % conditional echo
                              %                                09347
                              REPEAT CASE 2 (-1); % EXIT
                              routine %                        09348
                              END;                             09349
               ENDCASE;                                        09350
               % fall through if not a delimiter character %
                                                               09351
                  REPEAT CASE (0);                             09352
               END;                                            09353
            =CD:                                               09354
               SIGNAL (cmdelete);                              09355
            =BC:                                               09356
               BEGIN                                           09357
               IF fullflag AND (recogflag = 0) THEN            09358
                  BEGIN                                        09359
                  fullflag _ FALSE;                            09360
                  cmode.llcmd _ TRUE;                          09361
                  recmode _ recogmode;                         09362
                  recogflag _ 0;                               09363
                  END                                          09364
               ELSE                                            09365
```

```
                BEGIN                                                   09366
                recogflag _ MAX(0,recogflag-1);                         09367
                fbctl( nextchar );                                      09368
                END;                                                    09369
             REPEAT CASE;                                               09370
             END;                                                       09371
          =BW:                                                          09372
             BEGIN                                                      09373
             recogflag _ FALSE;                                         09374
             fullflag _ FALSE;                                          09375
             cmode.llcmd _ TRUE;                                        09376
             recmode _ recogmode;                                       09377
             fbctl( nextchar );                                         09378
             REPEAT CASE;                                               09379
             END;                                                       09380
          ='?:        % request for subhelp %                           09381
             BEGIN                                                      09382
             fbhelp( &curptr, cmode);                                   09383
             REPEAT CASE;                                               09384
             END;                                                       09385
          = 'S-100B:  %  <^S> request for syntax subhelp %
                                                                        09386
             BEGIN                                                      09387
             cshelp( &curptr, cmode, TRUE);                             09388
             REPEAT CASE;                                               09389
             END;                                                       09390
          = ALT,                                                        09391
          = $eschar: % possible right delimiter %                       09392
             BEGIN                                                      09393
             kwrstate _ 1;                                              09394
             % check for a right delimiter %                            09395
                IF recmode = mdemand AND recogflag AND
                fbstr.L # 0                                             09396
                   THEN                                                 09397
                      BEGIN                                             09398
                      % EXIT the routine %                              09399
                         REPEAT CASE  (-1);                             09400
                      END;                                              09401
             % fall through to ENDCASE %                                09402
                REPEAT CASE (0);                                        09403
             END;                                                       09404
          = -1:       % common routine exit point %                     09405
             BEGIN                                                      09406
             kwrstate _ 1;                                              09407
             % set ptrx to value set up by nextkey %                    09408
                ptrx _ work.savex;                                      09409
                ptrstk[ptrx-1] _ work.savstk;                           09410
             resultptr _ hindex;                                        09411
             IF hindex.LH # 0                                           09412
                THEN resultptr _ hindex.LH;                             09413
             resultptr[1] _ hindex.RH;                                  09414
             % save input chars for possible repeat cmd %
                                                                        09415
                IF keysaveflag THEN                                     09416
                   BEGIN                                                09417
                   IF fbstr.L > 0 THEN                                  09418
```

```
                        BEGIN                                   09419
                        IF fullflag % excape from expert %
                                                                09420
                           THEN *keyinpstr* _ *keyinpstr*, SP;
                                                                09421
                        *keyinpstr* _ *keyinpstr*, *fbstr*;
                                                                09422
                        END;                                    09423
                     IF (recmode = mexpert AND NOT fullflag)
                                                                09424
                        OR recmode = mdemand THEN               09425
                           *keyinpstr* _ *keyinpstr*, nextchar;
                                                                09426
                     END;                                       09427
                  % echo the keyword string %                   09428
                     CASE echotype OF                           09429
                        = 1: % full feedback %                  09430
                           fbctl( keyword, hindex.RH );         09431
                        = 2: % conditional feedback %           09432
                           IF nlmode = fulldisplay              09433
                              OR fbackmode = verbsmode          09434
                                 THEN REPEAT CASE(1)            09435
                                 ELSE fbctl( addchar, SP);      09436
                     ENDCASE;                                   09437
                  IF msjfn THEN  %monitoring commands%          09438
                     msrecord (mcomword, resultptr[1], 0);      09439
                  kwrstate _ -1;                                09440
                  RETURN( &resultptr );                         09441
                  END;                                          09442
               ENDCASE                                          09443
                  BEGIN                                         09444
                  kwrstate _ 1;                                 09445
                  % initialize the keyword search %             09446
                     ptrx _ oldptrx;                            09447
                     keyinit( &curptr, $work, cmode, FALSE );
                                                                09448
                     IF cmode.11cmd                             09449
                        THEN                                    09450
                           BEGIN % single character recognition mode
                           %                                    09451
                           IF hindex _ nextkey( $work, nextchar )
                                                                09452
                              THEN                              09453
                                 BEGIN                          09454
                                 REPEAT CASE (-1); % EXIT %     09455
                                 END                            09456
                              ELSE                              09457
                                 % didn't find anything appropriate %
                                                                09458
                                 IF nofail THEN                 09459
                                     % put up ? and try again % 09460
                                    BEGIN                       09461
                                    IF msjfn THEN  %monitoring
                                    commands%                   09462
                                       msrecord(mbell, 0, 0);   09463
                                    fbctl( '? );                09464
```

```
                        REPEAT CASE;                    09465
                        END;                            09466
                    END                                 09467
                ELSE                                    09468
                    BEGIN % multiple character recognition
                    mode %                              09469
                    *fbstr* _ *fbstr*, nextchar;        09470
                    hits _ 0;                           09471
                    WHILE (hitx _ nextkey( $work, nextchar ))
                    # 0 DO                              09472
                        IF (hits _ hits + 1) = 1 THEN   09473
                            BEGIN                       09474
                            hindex _ hitx;              09475
                            fbctl( addchar, nextchar);  09476
                            END                         09477
                        ELSE REPEAT CASE;               09478
                    % hits is the count of partial matches %
                                                        09479
                        CASE hits OF                    09480
                            =1: % have found the keyword % 09481
                                BEGIN                   09482
                                BUMP recogflag;         09483
                                CASE recmode OF         09484
                                    = mexpert,          09485
                                    = manticipatory:    09486
                                        BEGIN           09487
                                        REPEAT CASE 3 (-1); % exit
                                        %               09488
                                        END;            09489
                                    = mfixed:           09490
                                        IF fbstr.L = fixl OR
                                        fbstr.L = [hindex.RH].L
                                        THEN            09491
                                        REPEAT CASE
                                        (manticipatory); 09492
                                ENDCASE;                09493
                                REPEAT CASE 2;          09494
                                END;                    09495
                            =0: % no matches %          09496
                                BEGIN                   09497
                                fbstr.L _ fbstr.L-1; % get rid of
                                the bum char from the feedback
                                buffer %                09498
                                IF msjfn THEN  %monitoring
                                commands%               09499
                                    msrecord(mbell, 0, 0); 09500
                                fbctl( '? );            09501
                                END;                    09502
                            ENDCASE;                    09503
                        IF nofail THEN REPEAT CASE;     09504
                    END;                                09505
            END;                                        09506
        ptrx _ oldptrx;                                 09507
        kwrstate _ FALSE;                               09508
        RETURN (0); % FAILURE EXIT %                    09509
    END;                                                09510
```

```
        = backup:                                                       09511
           BEGIN                                                        09512
           IF msjfn THEN  %monitoring commands%                         09513
              msrecord(mcombackup, resultptr[1], 0);                    09514
           REPEAT CASE (cleanup);                                       09515
           END;                                                         09516
        = cleanup:                                                      09517
           BEGIN                                                        09518
           % reset feedback state information %                         09519
              cflstr.L _ resultptr.fblen;                               09520
           % reset keyword input save info (for possible repeat cmd)
           %                                                            09521
              keyinpstr.L _ resultptr.keyinlength;                      09522
              keysaveflag _ resultptr.ksaveflag;                        09523
           END;                                                         09524
     ENDCASE;                                                           09525
     RETURN;                                                            09526
     END.

                                                                        09527
% RECOGNITION SUPPORT ROUTINES %                                       02126
  (vocab) PROC( index, astr ); %returns vocabulary string, given
  index and address of astring %                                        02127
     LOCAL                                                              02128
        fflag,                                                          05171
        bp, % byte pointer %                                            02129
        count,      % character count in vocab string %                  02130
        char;       % next char in vocabulary string %                   02131
     REF astr;                                                          02132
     %---------------------%                                           02133
     *astr* _ NULL;                                                     02134
     count _ 0;                                                         02135
     bp _ chbmty + index;                                               02136
     UNTIL (char _ ^bp) = 0 DO                                          02137
        BEGIN                                                           02138
        IF (count _ count + 1) # 1 THEN                                 02139
           BEGIN                                                        05172
           IF fflag THEN                                                05175
              CASE char OF                                              02140
                 IN ["A, "Z]:                                           02141
                    char _ char + 40B; % force to lower case %          02142
              ENDCASE;                                                  02143
           END                                                          05173
        ELSE fflag _ IF char = "< THEN FALSE ELSE TRUE;                 05174
        *astr* _ *astr*, char;                                          02144
        END;                                                            02145
     RETURN;                                                            02146
     END.

                                                                        02147
  (nextkey) PROCEDURE( work, nextchar );                                02148
     LOCAL                                                              02149
        bp, % byte pointer %                                            02150
        bp2,        % byte pointer %                                    02151
        cmode,      % local copy of work.currcm %                       02152
        curptr,     % ptr to current path stack %                       02153
        i,          % index variable for compare loop %                 02154
        instptr,    % ptr to current instruction word %                 02155
```

```
        l,          % current length of recognition string %    02156
        proc,       % $pfcall function address %                 02157
        result,     % return result %                            02158
        tempptr,    % temp inst. ptr %                           02159
        value;      % compare value for multi char search %      02160
    REF                                                         02161
        curptr,                                                 02162
        instptr,                                                02163
        proc,                                                   02164
        tempptr,                                                02165
        work;                                                   02166
    REF fbstr, inpt;                                            05169
    REF tda, sysmsg, hlpcmdstk;                                 05170
    %---------------------%                                     02167
    % set up some state info (change later) %                   02168
        cmode _ work.currcm;                                    02169
        result _ 0;                                             02170
        l _ fbstr.L;                                            02171
    % chase down the data structures looking for a match %      02172
        WHILE (&instptr _ work.currinst) # 0 AND result = 0 DO  02173
            BEGIN                                               02174
            % decrement work.optcount if endopt was set on the last
            time through here %                                 02175
            IF work.endopt                                      02176
                THEN                                            02177
                    BEGIN                                       02178
                    work.optcount _ work.optcount - 1;          02179
                    work.endopt _ FALSE;                        02180
                    END;                                        02181
            CASE instptr.opcode OF                              02182
                = $keyop:                                       02183
                    BEGIN                                       02184
                    IF work.optflag AND work.optcount = 0       02185
                        THEN EXIT CASE;                         02186
                    IF instptr.ctrl .A cmode = cmode            02187
                        THEN                                    02188
                            IF cmode.l1cmd THEN                 02189
                                BEGIN                           02190
                                bp _ chbmty + instptr.addr;     02191
                                IF nextchar < 0 OR ^bp = nextchar 02192
                                    THEN REPEAT CASE (-1);      02193
                                END                             02194
                            ELSE                                02195
                                BEGIN                           02196
                                % if in secondary expert mode dont check
                                L1 cmds%                        02197
                                    IF instptr.ctrl.l1cmd AND work.usrmode
                                    = mexpert
                                     THEN EXIT CASE;            02198
                                % exit if string is empty %     02199
                                    IF l = 0 THEN REPEAT CASE (-1); 02200
                                bp2 _ chbmty + &fbstr;          02201
                                bp _ chbmty + instptr.addr;     02202
                                IF l <= 5 THEN % do fast compare % 02203
                                    BEGIN                       02204
                                    bp.bpsize _ bp2.bpsize _ 7 * l; 02205
```

```
                        value _ ^bp2;                                   02206
                        IF ^bp = value THEN REPEAT CASE (-1);           02207
                        END                                             02208
                     ELSE                                               02209
                        BEGIN % compare each character %                02210
                        FOR i _ 1 UP UNTIL > 1 DO                       02211
                           IF ^bp # ^bp2 THEN EXIT CASE; % no
                           hit %                                        02212
                           REPEAT CASE (-1); % hit %                    02213
                        END;                                            02214
                     END;                                               02215
      END;                                                              02216
   = $xecute:                                                           02217
      BEGIN                                                             02218
      % stack ptr to the instruction and repeat the case
      with the invoked tree as our next alternative %
                                                                        02219
         ptrstk[ ptrx ] _ &instptr;                                     02220
         BUMP ptrx;                                                     02221
         IF (work.nextx _ ptrx) > $ptrssize                             02222
            THEN SIGNAL (interperr, $"ptrstk
            overflowed");                                               02223
         &instptr _ work.currinst _ instptr.addr;                       02224
         REPEAT CASE;                                                   02225
      END;                                                              02226
   = $option,                                                           02227
   = $anyof:                                                            02228
      IF nextchar < 0 % for subhelp purposes only %                     02229
      OR work.optflag % recognition of optional keywords
      %                                                                 02230
         THEN                                                           02231
            BEGIN                                                       02232
            work.optcount _ work.optcount + 1;                          02233
            work.firstinst _ &instptr;                                  02234
            REPEAT CASE  ($xecute);                                     02235
            END;                                                        02236
   = -1: % come here when we've found a partial match %
                                                                        02237
      BEGIN                                                             02238
      &curptr _ work.currpath;                                          02239
      curptr.curnodeptr _                                               02240
         IF work.optcount > 0                                           02241
            THEN work.firstinst                                         02242
            ELSE &instptr;                                              02243
      result _ instptr.addr;                                            02244
      result.LH _ instptr.val2;                                         02245
      work.savex _ ptrx;                                                02246
      work.savstk _ ptrstk[ptrx-1];                                     02247
      END;                                                              02248
   = $confirm:                                                          02249
      IF nextchar < 0                                                   02250
         THEN result _ $"<OK>";                                         02251
   = $ssel,                                                             02252
   = $lsel,                                                             02253
   = $dsel:                                                             02254
```

```
            IF nextchar < 0                                          02255
               THEN result _ $"<CONTENT>";                           02256
         = $vwspecs:                                                 02257
            IF nextchar < 0                                          02258
               THEN result _ $"<VIEWSPECS>";                         02259
         = $levadj:                                                  02260
            IF nextchar < 0                                          02261
               THEN result _ $"<LEVEL-ADJUST>";                      02262
         = $pfcall:                                                  02263
            IF nextchar < 0 THEN                                     02264
               BEGIN                                                 02265
               &curptr _ work.currpath;                              02266
               &proc _ instptr.addr;                                 02267
               *savhelpstr* _ NULL;                                  02268
               % $call the proc in "parsehelp" mode to find out
               what it is doing %                                    02269
                  proc( &curptr, parsehelp, $savhelpstr );
                                                                     02270
               result _ $savhelpstr;                                 02271
               END;                                                  02272
      ENDCASE                                                        02273
         IF nextchar < 0 THEN                                        02274
            BEGIN % look for some printable "subhelp"
            information %                                            02275
            &tempptr _ instptr.nsuccessor;                           02276
            WHILE &tempptr # 0 DO                                    02277
               CASE tempptr.opcode OF                                02278
                  = $keyop,                                          02279
                  = $xecute,                                         02280
                  = $option,                                         02281
                  = $anyof:                                          02282
                     BEGIN                                           02283
                     &instptr _ &tempptr;                            02284
                     REPEAT CASE 2;                                  02285
                     END;                                            02286
                  IN [$confirm, $levadj]:                            02287
                     REPEAT CASE 2 (tempptr.opcode);                 02288
               ENDCASE                                               02289
                  &tempptr _ tempptr.nsuccessor;                     02290
            END;                                                     02291
      % chase down alternatives to current instruction, if
      there are no more alternatives, then the ptrstk is popped
      to get a next instruction %                                    02292
         CASE &tempptr _ instptr.alternative OF                      02293
            = 0:       % null alternative chain %                    02294
               IF work.nextx > work.firstx THEN                      02295
                  BEGIN                                              02296
                  BUMP DOWN ptrx;                                    02297
                  &instptr _ ptrstk[ work.nextx _ ptrx];             02298
                  % if leaving an optional construct, then set
                  the $option flag %                                 02299
                     CASE instptr.opcode OF                          02300
                        = $option,                                   02301
                        = $anyof:                                    02302
                           work.endopt _ TRUE;                       02303
                     ENDCASE;                                        02304
```

```
                        REPEAT CASE;                                    02305
                        END;                                            02306
                    = &instptr:        % check for a looping construct
                    %                                                   02307
                        REPEAT CASE (&tempptr _ 0);                     02308
                ENDCASE;                                                02309
            work.currinst _ &instptr _ &tempptr;                        02310
            END;                                                        02311
        RETURN (result);                                                02312
        END.
                                                                        02313
    (keyinit) PROCEDURE( curptr, work, lcmdmode, mode);                 05251
        % keyinit sets up a static work record for use by nextkey to
        sequence through the alternative keyword lists %                05252
        LOCAL instptr;                                                  05253
        REF instptr, curptr, work;                                      05254
        REF fbstr, inpt;                                                05255
        REF tda, sysmsg, hlpcmdstk;                                     05256
        %----------------------%                                        05257
        % set up some state info (change later) %                       05258
            &instptr _ curptr.begnodeptr;                               05259
            IF NOT mode AND instptr.opcode IN [$option, $anyof]         05260
                THEN &instptr _ instptr.addr;                           05261
            work.firstinst _ work.currinst _ &instptr;                  05262
            work.currcm _ lcmdmode;                                     05263
            work.optflag _ FALSE;                                       05264
            work.endopt _ FALSE;                                        05265
            work.optcount _ 0;                                          05266
            work.firstx _ work.nextx _ ptrx;                            05267
            work.currpath _ &curptr;                                    05268
            work.usrmode _                                              05269
                (IF lcmdmode.llcmd THEN recogmode ELSE recog2mode);     05270
        RETURN;                                                         05271
        END.
                                                                        05272
% FEEDBACK CONTROL ROUTINES %                                           02334
    (fbctl) PROCEDURE( function, p1 ); % feedback control routine %
                                                                        07486
        % user feedback operations are performed here,  above this
        level there should not be much need for knowing whether the
        user is in TNLS or DNLS %                                       07487
        LOCAL                                                           07488
            ls, % ptr to link stack entry %                             07489
            ls1,        % ptr to link stack entry %                     07490
            char,       % character value %                            07491
            l,          % length count %                               07492
            norstsw;    %prevent extra screen refresh in addcalit%      08318
        LOCAL STRING tempstr[100];                                      07493
        % REF VARIABLES %                                              07494
            REF fbstr, inpt;                                            07495
            REF tda, sysmsg, hlpcmdstk;                                 07496
            REF                                                         07497
                ls1,    % ptr to link stack entry %                    07498
                ls;     % ptr to link stack entry %                    07499
        %------------------------%                                      07500
        l _ fbstr.L;                                                    07501
```

```
IF nlmode = fulldisplay THEN                                    07502
   BEGIN % DNLS %                                               07503
   norstsw _ FALSE;   %set to TRUE only in norstcalit case%
                                                                08319
   CASE function OF                                             07504
      =clearcfl:                % clear feedback buffer %      07505
         BEGIN                                                  07506
         *fbstr* _ NULL;                                        07507
         cflstr.L _ cflpos _ 0;                                 07508
         cfldsp();                                              07509
         disarm();                                              07510
         END;                                                   07511
      =addchar:          % add character to feedback buffer %
                                                                07512
         BEGIN                                                  07513
         IF 1 > 1                                               07514
            THEN DSP(...*fbstr* )                               07515
            ELSE DSP( *fbstr* );                                07516
         END;                                                   07517
      =keyword:          % feedback keyword %                   07518
         BEGIN                                                  07519
         vocab( p1, &fbstr );                                   07520
         IF 1 > 0                                               07521
            THEN DSP(...*fbstr* )                               07522
            ELSE DSP( *fbstr* );                                07523
         *fbstr* _ NULL;                                        07524
         IF fbackmode = tersemode THEN                          07525
            DSP(^);                                             07526
         END;                                                   07527
      =startrec:         % begin keyword recognition %          07528
         BEGIN                                                  07529
         DSP(^);                                                07530
         END;                                                   07531
      =BC:               % backspace character %                07532
         IF 1 <= 0 THEN                                         07533
            REPEAT CASE (BW)                                    07534
         ELSE                                                   07535
            BEGIN % delete last character ffrom string %        07536
            fbstr.L _ 1 - 1;                                    07537
            DSP(...*fbstr*  );                                  07538
            END;                                                07539
      =BW:               % backspace word %                    07540
         IF 1 > 0 THEN                                          07541
            BEGIN                                               07542
            *fbstr* _ NULL;                                     07543
            cflstr.L _ MAX( 0, cflpos-1 );                      07544
            cfldsp();                                           07545
            END                                                 07546
         ELSE                                                   07547
            IF inhelp THEN                                      07548
               % Do NOT permit backspacing out in this case. %
                                                                07549
               REPEAT CASE('?)                                  07550
            ELSE SIGNAL (popstate); % back up to previous point
            in parse %                                          07551
      ='?:               % unrecognizable command %             07552
```

```
    BEGIN                                                          07553
    IF auxinflag % reading from aux. input source %                07554
       THEN auxinterminate();                                      07555
    qm(); % put out question mark %                                07556
    lookc();      % wait for next character %                     07557
    END;                                                           07558
=echostr:         % echo supplied string in cfl %                 07559
    IF fbackmode = verbsmode THEN                                  07560
       BEGIN                                                       07561
       *tempstr* _ '(, *[p1]*, ');                                 07562
       DSP( *tempstr*  );                                          07563
       END;                                                        07564
=rechostr:        % replace previously supplied string in
cfl %                                                              07565
    BEGIN                                                          07566
    *tempstr* _ '(, *[p1]*, ');                                    07567
    DSP( ...*tempstr*  );                                          07568
    END;                                                           07569
=fbendlit:        % append to lit str and wait for input %
                                                                   07570
    BEGIN                                                          07571
    aplit( p1 );                                                   07572
    lookc(); % wait for user to type in next char %                07573
    rstlit();                                                      07574
    END;                                                           07575
=typelit:         % put up literal string (not in cfl) %
                                                                   07576
    BEGIN                                                          07577
    litdpy( p1 );                                                  07578
    lookc(); % wait for user to type in next char %                07579
    rstlit();                                                      07580
    END;                                                           07581
=typenulllit:     % put up null lit string (not in cfl) %
                                                                   07582
    BEGIN                                                          07583
    litdpy( $"" );                                                 07584
    END;                                                           07585
=typecalit:       % put up literal string (not in cfl) %
                                                                   07586
    BEGIN                                                          07587
    litdpy( p1 );                                                  07588
    REPEAT CASE(addcalit);                                         07589
    END;                                                           07590
=norstcalit:      % put up literal string (not in cfl) %
                                                                   08320
    BEGIN                                                          08321
    litdpy( p1 );                                                  08322
    norstsw _ TRUE;                                                08323
    REPEAT CASE(addcalit);                                         08324
    END;                                                           08325
=addcalit:        % put up literal string (not in cfl) %
                                                                   07591
    BEGIN                                                          07592
    aplit($"

    Type <OK> to continue.");                                      07593
```

```
            % wait for user to type in CA or CD %                    07594
               CASE inpt() OF                                        07595
                  = CA: IF norstsw THEN litline _ 0 ELSE rstlit();
                                                                     07596
                  = CD:                                              07597
                     BEGIN                                           07598
                     rstlit();                                       07599
                     SIGNAL(statesig);                               07600
                     END;                                            07601
                  ENDCASE REPEAT CASE;                               07602
               END;                                                  07603
            =startparams:    % begin parameter collection %          07604
               BEGIN                                                 07605
               IF NOT &tda THEN &tda _ lda();        % for tabs %
                                                                     07606
               IF fbackmode = tersemode                              07607
                  THEN af();                                         07608
               END;                                                  07609
            =incues: % provide inpt prompts %                        07610
               IF inprompt # noprompts THEN                          07611
                  BEGIN                                              07612
                  *cflarw* _ " > ", *[p1]*;                          07613
                  arowon _ TRUE;                                     07614
                  mrk();                                             07615
                  END;                                               07616
            =fbpop:          % cleanup after popping states %        07617
               BEGIN                                                 07618
               cueflg _ FALSE;                                       07619
               cfldsp();                                             07620
               END;                                                  07621
            =fbaddlit:       % add p1 to literal area %              07622
               BEGIN                                                 07623
               aplit( p1 );                                          07624
               IF nldevice = devlproc THEN track();                  07625
               END;                                                  07626
            ENDCASE err($"Illegal call to fbctl");                   07627
         END                                                         07628
      ELSE                                                           07629
         BEGIN % TNLS %                                              07630
         CASE function OF                                            07631
            =clearcfl:                  % clear feedback buffer %    07632
               BEGIN                                                 07633
               *fbstr* _ NULL;                                       07634
               !rfpos(100B);                                         09843
               IF NOT r2 OR r2.RH THEN crlf();                       07635
               END;                                                  07636
            =addchar:        % echo character from feedback buffer %
                                                                     07637
               BEGIN                                                 07638
               IF l > 1 THEN                                         07639
                  CASE p1 OF                                         07640
                     IN ['A, 'Z]:                                    07641
                        p1 _ p1 + 40B; % force to lower case %       07642
                  ENDCASE;                                           07643
               todco( p1 );                                          07644
               END;                                                  07645
```

```
=keyword:          % feedback keyword %                    07646
   BEGIN                                                  07647
   IF 1 < feedbk THEN                                     07648
      BEGIN                                               07649
      vocab( p1, &fbstr );                                07650
      *fbstr* _                                           07651
         *fbstr*[1+1 TO MIN( fbstr.L, feedbk)];           07652
      echo( &fbstr );                                     07653
      END;                                                07654
   todco( SP );                                           07655
   *fbstr* _ NULL;                                        07656
   END;                                                   07657
=startrec:         % begin keyword recognition %          07658
   NULL;                                                  07659
=BC:               % backspace character %                07660
   IF 1 <= 0 THEN                                         07661
      REPEAT CASE (BW)                                    07662
   ELSE                                                   07663
      BEGIN % delete last character ffrom string %        07664
      todco( '\ );                                        07665
      todco( *fbstr*[ 1 ] );                              07666
      fbstr.L _ 1 - 1;                                    07667
      END;                                                07668
=BW:               % backspace word %                     07669
   IF 1 > 0 THEN                                          07670
      BEGIN                                               07671
      *fbstr* _ NULL;                                     07672
      todco( '_ );                                        07673
      END                                                 07674
   ELSE                                                   07675
      IF cutstop THEN                                     07676
         % Do NOT permit backspacing out in this case. %
                                                          07677
         REPEAT CASE('?)                                  07678
      ELSE SIGNAL (popstate); % back up to previous point
      in parse %                                          07679
='?:               % unrecognizable command %             07680
   BEGIN                                                  07681
   IF auxinflag % reading from aux. input source %        07682
      THEN auxinterminate();                              07683
   typech( 7B ); % ? %                                    07684
   END;                                                   07685
=echostr:          % echo supplied string in cfl %        07686
   IF fbackmode = verbsmode THEN                          07687
      BEGIN                                               07688
      *tempstr* _ '(,                                     07689
         *[p1]*[1 TO MIN([p1].L, feedbk)], ');            07690
      feedbk _ feedbk+2;                                  07691
      ON SIGNAL ELSE feedbk _ feedbk-2;                   07692
      echo( $tempstr );                                   07693
      feedbk _ feedbk-2;                                  07694
      ON SIGNAL ELSE;                                     07695
      todco( SP );                                        07696
      END;                                                07697
=rechostr:         % replace previously supplied string in
cfl %                                                     07698
```

```
            BEGIN                                                   07699
            *tempstr* _ "(, *[p1]*[1 TO MIN([p1].L, feedbk)], ");
                                                                    07700
            echo( $"_ " );                                          07701
            feedbk _ feedbk+2;                                      07702
            ON SIGNAL ELSE feedbk _ feedbk-2;                       07703
            echo( $tempstr );                                       07704
            feedbk _ feedbk-2;                                      07705
            ON SIGNAL ELSE;                                         07706
            END;                                                    07707
         =typelit, =typecalit, =norstcalit:       % put up literal
         string (not in cfl) %                                      07708
            BEGIN                                                   07709
            crlf();                                                 07710
            typeas( p1 );                                           07711
            END;                                                    07712
         =typenulllit:    % put up null lit string (not in cfl) %
                                                                    07713
            BEGIN                                                   07714
            crlf();                                                 07715
            END;                                                    07716
         =addcalit:       % put up literal string (not in cfl) %
                                                                    07717
            NULL;                                                   07718
            %---- don't need to do anything if device is
            typewriter ------%                                      07719
         =startparams:    % begin parameter collection %           07720
            BEGIN                                                   07721
            echoff();                                               07722
            END;                                                    07723
         =incues: % provide inpt prompts %                          07724
            IF inprompt # noprompts THEN                            07725
               IF NOT cueflg THEN                                   07726
                  BEGIN                                             07727
                  cueflg _ TRUE;                                    07728
                  typeas( p1 );                                     07729
                  typech( SP );                                     07730
                  END;                                              07731
         =fbpop:          % cleanup after popping states %         07732
            BEGIN                                                   07733
            cueflg _ FALSE;                                         07734
            echo( $" _ ");                                          07735
            END;                                                    07736
         =fbaddlit, =fbendlit:    % add p1 to literal area %        07737
            BEGIN                                                   07738
            typeas( p1 );                                           07739
            END;                                                    07740
         ENDCASE err($"Illegal call to fbctl");                     07741
      END;                                                          07742
   RETURN;                                                          07743
   END.
                                                                    07744
% HELP ROUTINES %                                                  02566
   (fbhelp) PROCEDURE( list, cmode ); % lists alternatives %        09528
      LOCAL                                                         09529
         svllcmd, svrec1, svrec2, ncols, hlpblk, achn, cinstptr,
```

```
                                                                    09530
    savecnptr,                                                      09531
    nextchar,    % next inpt character %                             09532
    hindex,      % hit index into vocab for keyword %                09533
    helpcnt,     % number of keywords in helpstr %                   09534
    % work SHOULD BE wasize (in PDATA) WORDS LONG %                  09535
       work[30];         % state buffer for keyinit and nextkey %
                                                                    09536
LOCAL TEXT POINTER ptr, t1;                                         09537
LOCAL STRING                                                        09538
    lev2str[10],         % level 2 command promtp %                 09539
    keywordstr[25],      % contains keywordstr %                    09540
    helpstr[2000];       % contains subhelp info %                  09541
% REF VARIABLES %                                                   09542
    REF fbstr, inpt, cinstptr;                                      09543
    REF tda, litda, sysmsg, hlpcmdstk;                              09544
    REF list;                                                       09545
%--------------------%                                             09546
IF msjfn THEN  %monitoring commands%                                09547
    msrecord(mquestion, 0, 0);                                      09548
% Do special things in or at Help command. %                       09549
    IF inhelp OR athelp THEN                                        09550
       BEGIN                                                        09551
       *helpstr* _ "Current alternatives are:";                     09816
       ap2clstr($helpstr, IF nlmode=fulldisplay THEN $"your OK
       button  (OK is <CTRL-D>)" ELSE $"your OK button  (OK is
       <CTRL-D> and usually a Carriage Return)");                   09817
       ap2clstr($helpstr, $"a word or number (to see its
       description) followed by your OK");                          09818
       IF inhelp THEN                                               09824
          BEGIN                                                     09825
          ap2clstr($helpstr, $"Type  <  to see previous
          views.");                                                 09819
          ap2clstr($helpstr, $"Type  ^  to see your
          surroundings.");                                          09820
          END;                                                      09826
       ap2clstr($helpstr, $"Command Delete <CTRL-X> returns you
       to where you were before you entered Help.");                09821
       ap2clstr($helpstr, $"Hit <CTRL-Q> for help with Help.");
                                                                    09822
       IF nlmode#fulldisplay THEN ap2clstr($helpstr, $"(Help)
       </T: ");                                                     09823
       fbctl( typelit, $helpstr);                                   09563
       RETURN;                                                      09564
       END;                                                         09565
% initialize %                                                      09581
    *lev2str* _ "<>";                                               09582
    ncols _ (litda.daright - litda.daleft) / litda.dahinc;          09583
       ncols _ IF nlmode = fulldisplay THEN ncols/3 ELSE
       tda.damcol/3;                                                09584
    svrec1 _ recogmode;                                             09585
    svrec2 _ recog2mode;                                            09586
    svllcmd _ FALSE;                                                09587
    hlpblk _ achn _ 0;                                              09588
% save current node pointer in path stack %                        09589
    savecnptr _ list.curnodeptr;                                    09590
```

```
    % cleanup on any errors %                                          09591
      ON SIGNAL ELSE                                                   09592
         BEGIN                                                         09593
         ON SIGNAL ELSE;                                               09594
         IF hlpblk THEN dlfmb( hlpblk := 0);                           09595
         IF svllcmd AND svrec1 = mexpert THEN                          09596
            BEGIN                                                      09597
            recogmode _ svrec1;                                        09598
            recog2mode _ svrec2;                                       09599
            END;                                                       09600
         savecnptr _ list.curnodeptr;                                  09601
         END;                                                          09602
    % diddle around to get all possibilites %                          09603
      svllcmd _ cmode.llcmd := FALSE;                                  09604
      IF svllcmd AND recogmode = mexpert THEN                          09605
         recogmode _ recog2mode _ manticipatory;                       09606
    % get storage for building strings %                               09607
      IF NOT hlpblk _ dlgmb( 400 ) THEN GOTO fbrhelp;                  09608
    % set up some state info (change later) %                          09609
      % *** note:  we are using a new work area to control the
      sequencing through the keyword list, but we still use the
      fbstr from the work area presently in effect %                   09610
         *helpstr* _ "Current Alternatives are:", CR, LF;              09611
         helpcnt _ 0; %number of words in helpstr %                    09612
         nextchar _ -1; % so nextkey will give back next thing %
                                                                       09613
         keyinit( &list, $work, cmode, TRUE);                         09614
         &cinstptr _ work.currinst;                                    09615
    % use nextkey to get all keywords on current list and add them
    to the helpstring %                                                09616
      WHILE (hindex _ nextkey( $work, nextchar )) # 0 DO               09617
         BEGIN                                                         09618
         IF inptrf THEN EXIT LOOP;                                     09619
         IF NOT                                                        09620
            addhelp( hindex.RH, hlpblk, $work, ncols, $lev2str,
            svllcmd, $achn, $helpcnt, svrec1, &cinstptr) THEN EXIT
            LOOP;                                                      09621
         BUMP helpcnt;                                                 09622
         &cinstptr _ work.currinst;                                    09623
         END;                                                          09624
    % output the helpstring %                                          09625
      fbbldhlp( $helpstr, achn, helpcnt, ncols);                       09626
    (fbrhelp):                                                         09627
    % restore current node pointer in pathst %                        09628
      list.curnodeptr _ savecnptr;                                     09629
    IF hlpblk THEN dlfmb( hlpblk := 0);                                09630
    IF svllcmd AND svrec1 = mexpert THEN                               09631
      BEGIN                                                            09632
      recogmode _ svrec1;                                              09633
      recog2mode _ svrec2;                                             09634
      END;                                                             09635
    RETURN;                                                            09636
    END.
                                                                       09637
  (ap2clstr) PROCEDURE (string, apstring);                             09810
    %append apstring to string separated by two CRLF's%                09815
```

```
    REF string, apstring;                                          09811
    *string* _ *string*, CR, LF, CR, LF, *apstring*;               09812
    RETURN;                                                        09813
    END.                                                           09814
(addhelp) PROCEDURE( index, hlpblk, work, ncols, lev2str, clvl,
achn, helpcnt, reclmd, cinstptr ); % edits name and adds to
helpstr %                                                          04648
    LOCAL                                                          04649
       ablk, temp, pscnt,                                          04650
       nsptr,      % ptr for getting noise words %                  04651
       char,       % first char in keywordstr %                     04652
       i,                                                          04653
       l;                                                          04654
    LOCAL TEXT POINTER tp1, tp2, tp3;                              04886
    LOCAL STRING                                                   04655
       blnkstr[10],                                                04656
       nsstr[100],                                                 04657
       keywordstr[100];    % vocabulary string %                   04658
    REF helpcnt, achn, ablk, lev2str, work, nsptr, cinstptr;       04659
    %-------------------%                                          04660
    % initialize balnk string %                                    04891
       *blnkstr* _ "          ";                                   04661
    % deal with ENTER and REPEAT Commands and ^Q %                 04892
       IF NOT helpcnt THEN                                         05923
          BEGIN                                                    05924
          IF NOT add2help($"<CTRL-Q>: HELP", hlpblk, &lev2str,
          &achn, ncols, clvl) THEN RETURN( FALSE );                05926
          BUMP helpcnt;                                            05927
          IF NOT add2help($"<CTRL-S>: SYNTAX", hlpblk, &lev2str,
          &achn, ncols, clvl) THEN RETURN( FALSE );                07110
          BUMP helpcnt;                                            07111
          IF basestateflag AND (NOT kwrstate) AND (clvl OR reclmd #
          mexpert) THEN                                            04662
             BEGIN                                                 04663
             IF  [$sbstack + sbstkx - $sbentsize].sbptr =
             $nlseditor THEN                                       04888
                BEGIN                                              04889
                IF NOT add2help($"<ENTER>", hlpblk, &lev2str,
                &achn, ncols, clvl) THEN RETURN( FALSE );          04664
                BUMP helpcnt;                                      04665
                END;                                               04890
             IF NOT add2help($"<REPEAT>", hlpblk, &lev2str, &achn,
             ncols, clvl) THEN RETURN( FALSE );                    04666
             BUMP helpcnt;                                         04667
             END;                                                  04668
          END;                                                     05925
    % take care of parse function prompts %                       04878
       IF cinstptr.opcode = $pfcall THEN                           04881
          BEGIN                                                    04882
          *keywordstr* _ NULL;                                     04910
          % call the parsefunction to get string %                 04915
             [cinstptr.addr]( 0, parseqmark, $keywordstr);         04893
          % find and handle first entity %                        04916
             *nsstr* _ 0;                                          04905
             FIND SF(*keywordstr*) ^tp1                            04894
                [ *nsstr* ^tp3 ^tp2 _tp2 / ENDCHR ^tp3 ^tp2];      04906
```

```
            *nsstr* _ tp1 tp2;                                      04896
            IF nsstr.L THEN                                         04911
               BEGIN                                                04917
               IF NOT add2help($nsstr, hlpblk, &lev2str, &achn,
               ncols, clvl) THEN RETURN( FALSE );                   04895
               END                                                  04918
            ELSE RETURN( TRUE );                                    04919
         % deal with the remaining (if any) entities %              04920
            LOOP                                                    04897
               BEGIN                                                04898
               *nsstr* _ 0;                                         04907
               FIND tp3 > ^tp1                                      04899
                  [ *nsstr* ^tp3 ^tp2 _tp2 / ENDCHR ^tp3 ^tp2];
                                                                    04921
               *nsstr* _ tp1 tp2;                                   04900
               IF nsstr.L THEN                                      04922
                  BEGIN                                             04924
                  IF NOT add2help($nsstr, hlpblk, &lev2str, &achn,
                  ncols, clvl) THEN RETURN( FALSE );                04901
                  END                                               04925
               ELSE EXIT LOOP;                                      04923
               BUMP helpcnt;                                        04902
               END;                                                 04904
         RETURN( TRUE );                                            04903
         END;                                                       04885
      % get a block for this entry %                                04669
       IF NOT &ablk _                                               04670
          dlgtblk( hlpblk, ((ncols+4)/5)+1+$fbhlpl ) THEN RETURN(
          FALSE );                                                  04671
       ablk.hbastr _ &ablk+$fbhlpl;                                 04672
       [ablk.hbastr].M _ ncols;                                     04673
      % get symbolic represenation for the keyword %               04674
       [$keywordstr+1] _ 0;                                         04675
       vocab( index, $keywordstr );                                 04676
      % get value of this keyword for sorting %                    04677
       ablk.hbval _ [$keywordstr+1] / 2;                            04678
      % get print representation for non-printing symbols (if any) %
                                                                    04679
       IF keywordstr.L = 1                                          04680
          AND (char _ *keywordstr*[1]) < 40B % non printing chars %
                                                                    04681
            THEN *keywordstr* _ *[ npstrad( char ) ]*;              04682
      % test for optional keywords %                               04683
       IF work.optcount AND NOT work.optflag                        04684
          THEN *keywordstr* _ '[, *keywordstr*, ']';                04685
      % take care of level 2 commands %                            04686
       IF clvl THEN                                                 04687
          BEGIN                                                     04688
          &nsptr _ work.currpath;                                   04689
          &nsptr _ nsptr.curnodeptr;                                04690
          IF nsptr.opcode = $keyop AND NOT nsptr.ctrl.llcmd THEN
                                                                    04691
            *keywordstr* _ *lev2str*, *keywordstr*                  04692
          ELSE                                                      04693
            *keywordstr* _ *blnkstr*[1 TO lev2str.L],
            *keywordstr*;                                           04694
```

```
        END;                                                    04695
    % edit in noise words %                                     04696
      &nsptr _ work.currpath;                                   04697
      &nsptr _ nsptr.curnodeptr;                                04698
      *nsstr* _ NULL;                                           04699
      WHILE &nsptr _ nsptr.nsuccessor DO                        04700
        IF nsptr.opcode # $necho THEN EXIT LOOP                 04701
        ELSE                                                    04702
          IF nsstr.L THEN                                       04703
            *nsstr* _ *nsstr*, SP, *[nsptr.addr]*               04704
          ELSE                                                  04705
            *nsstr* _ *[nsptr.addr]*;                           04706
    % edit the string %                                         04707
      temp _ ncols - keywordstr.L - 4;                          04708
      IF nsstr.L > temp THEN                                    04709
        *nsstr* _ *nsstr*[1 TO (temp-3)], "...";                04710
      IF nsstr.L THEN                                           04711
        *[ablk.hbastr]* _ *keywordstr*, " (", *nsstr*, ')       04712
      ELSE                                                      04713
        *[ablk.hbastr]* _ *keywordstr*;                         04714
    % put this block where it belongs %                         04715
      IF NOT (&nsptr _ achn.hbfor) THEN achn.hbfor _ &ablk      04716
      ELSE                                                      04717
        IF ablk.hbval < nsptr.hbval THEN % special for first one
        %                                                       04718
          BEGIN                                                 04719
          ablk.hbfor _ &nsptr;                                  04720
          achn.hbfor _ nsptr.hbbck _ &ablk;                     04721
          END                                                   04722
        ELSE                                                    04723
          CASE TRUE OF                                          04724
            ENDCASE                                             04725
              IF NOT nsptr.hbfor THEN                           04726
                BEGIN                                           04727
                nsptr.hbfor _ &ablk;                            04728
                ablk.hbbck _ &nsptr;                            04729
                END                                             04730
              ELSE                                              04731
                IF ablk.hbval < [nsptr.hbfor].hbval THEN
                                                                04732
                  BEGIN                                         04733
                  nsptr.hbfor.hbbck _ &ablk;                    04734
                  ablk.hbfor _ nsptr.hbfor;                     04735
                  ablk.hbbck _ &nsptr;                          04736
                  nsptr.hbfor _ &ablk;                          04737
                  END                                           04738
                ELSE                                            04739
                  BEGIN                                         04740
                  &nsptr _ nsptr.hbfor;                         04741
                  REPEAT CASE;                                  04742
                  END;                                          04743
    RETURN( TRUE );                                             04744
    END.
                                                                04745
(fbbldhlp) PROCEDURE ( helpstr, achn, helpcnt, ncols);          04599
    LOCAL nrows, i, j, temp, tptr;                              04600
```

```
    LOCAL STRING blnkstr[40];                                          04601
    REF helpstr, achn, tptr;                                           04602
    nrows _ (helpcnt+2)/3;                                             04603
    *blnkstr* _ "                                    ";                04604
    fbctl( typenulllit, );                                             04605
    fbctl( fbaddlit, &helpstr);                                        04606
    FOR i _ 0 UP UNTIL = nrows DO                                      04607
       IF &achn THEN                                                   04608
          BEGIN                                                        04609
          IF (inptrf := FALSE) THEN EXIT LOOP;                         04610
          *helpstr* _ NULL;                                            04611
          % do first one %                                             04612
             temp _ ncols - [achn.hbastr].L;                           04613
             *helpstr* _                                               04614
                *helpstr*, *[achn.hbastr]*, *blnkstr*[1 TO temp];
                                                                       04615
          % do second one %                                            04616
             &tptr _ &achn;                                            04617
             FOR j _ 1 UP UNTIL > nrows DO                             04618
                IF &tptr THEN &tptr _ tptr.hbfor                       04619
                ELSE EXIT LOOP;                                        04620
             IF &tptr THEN                                             04621
                BEGIN                                                  04622
                temp _ ncols - [tptr.hbastr].L;                        04623
                *helpstr* _                                            04624
                   *helpstr*, *[tptr.hbastr]*, *blnkstr*[1 TO
                   temp];                                              04625
                END;                                                   04626
          % do third one %                                             04627
             IF &tptr THEN                                             04628
                BEGIN                                                  04629
                FOR j _ 1 UP UNTIL > nrows DO                          04630
                   IF &tptr THEN &tptr _ tptr.hbfor                    04631
                   ELSE EXIT LOOP;                                     04632
                IF &tptr THEN                                          04633
                   *helpstr* _ *helpstr*, *[tptr.hbastr]*;             04634
                END;                                                   04635
          % finish the line %                                          04636
             *helpstr* _ *helpstr*, CR, LF;                            04637
          % give the user the line %                                   04638
             fbctl( fbaddlit, &helpstr );                              04639
          % go to the next line %                                      04640
             &achn _ achn.hbfor;                                       04641
          END                                                          04642
       ELSE EXIT LOOP;                                                 04643
    *helpstr* _ CR, LF, "---", *fbstr*;                                04644
    fbctl( fbendlit, &helpstr);                                        04645
    RETURN;                                                            04646
    END.
                                                                       04647
(add2help) PROCEDURE( kstring,  hlpblk, lev2str, achn, ncols, clvl
); % edits name and adds to helpstr %                                  04819
    LOCAL                                                              04820
       ablk, temp,                                                     04821
       nsptr,      % ptr for getting noise words %                      04822
       char,       % first char in keywordstr %                         04823
```

```
    1,                                                          04824
    1];                                                         04825
LOCAL STRING                                                    04826
    blnkstr[10],                                                04827
    nsstr[100],                                                 04828
    keywordstr[100];    % vocabulary string %                   04829
REF kstring, achn, ablk, lev2str, nsptr;                        04830
%--------------------%                                         04831
*blnkstr* _ "          ";                                       04832
% get a block for this entry %                                  04833
    IF NOT &ablk _                                              04834
       dlgtblk( hlpblk, ((ncols+4)/5)+1+$fbhlpl ) THEN RETURN(
       FALSE );                                                 04835
    ablk.hbastr _ &ablk+$fbhlpl;                                04836
    [ablk.hbastr].M _ ncols;                                    04837
% get symbolic represenation for the keyword %                  04838
    *keywordstr* _ *kstring*;                                   04839
% get value of this keyword for sorting %                       04840
    ablk.hbval _ 35M;                                           04841
% take care of level 2 commands %                               04842
    IF clvl THEN                                                04843
       *keywordstr* _ *blnkstr*[1 TO lev2str.L], *keywordstr*;
                                                                04844
% edit the string %                                            04845
    % Truncate first if length larger than ncols. %            07747
       IF keywordstr.L > ncols THEN keywordstr.L _ ncols;       07748
    *[ablk.hbastr]* _ *keywordstr*;                             04846
% put this block where it belongs %                            04847
    IF NOT (&nsptr _ achn.hbfor) THEN achn.hbfor _ &ablk        04848
    ELSE                                                        04849
       IF ablk.hbval < nsptr.hbval THEN % special for first one
       %                                                        04850
          BEGIN                                                 04851
          ablk.hbfor _ &nsptr;                                  04852
          achn.hbfor _ nsptr.hbbck _ &ablk;                     04853
          END                                                   04854
       ELSE                                                     04855
          CASE TRUE OF                                          04856
             ENDCASE                                            04857
                IF NOT nsptr.hbfor THEN                         04858
                   BEGIN                                        04859
                   nsptr.hbfor _ &ablk;                         04860
                   ablk.hbbck _ &nsptr;                         04861
                   END                                          04862
                ELSE                                            04863
                   IF ablk.hbval < [nsptr.hbfor].hbval THEN
                                                                04864
                      BEGIN                                     04865
                      nsptr.hbfor.hbbck _ &ablk;                04866
                      ablk.hbfor _ nsptr.hbfor;                 04867
                      ablk.hbbck _ &nsptr;                      04868
                      nsptr.hbfor _ &ablk;                      04869
                      END                                       04870
                   ELSE                                         04871
                      BEGIN                                     04872
                      &nsptr _ nsptr.hbfor;                     04873
```

```
                        REPEAT CASE;                                    04874
                        END;                                            04875
    RETURN( TRUE );                                                     04876
    END.
                                                                        04877
% <^S> syntax help routines %                                          06916
  (cshelp) PROCEDURE( list, cmode, fstrm ); % lists alternatives %
                                                                        09638
    LOCAL                                                               09639
      svllcmd, svrec1, svrec2, cinstptr,                                09640
      savecnptr, lookadr,                                               09641
      nextchar,   % next inpt character %                               09642
      hindex,     % hit index into vocab for keyword %                   09643
      % work SHOULD BE wasize (in PDATA) WORDS LONG %                   09644
        work[30];       % state buffer for keyinit and nextkey %
                                                                        09645
    LOCAL TEXT POINTER ptr, t1;                                         09646
    LOCAL STRING helpstr[300];                                          09647
    % REF VARIABLES %                                                  09648
      REF fbstr, inpt, cinstptr;                                        09649
      REF tda, litda, sysmsg, hlpcmdstk;                                09650
      REF list;                                                         09651
    %---------------------%                                            09652
    IF msjfn THEN  %monitoring commands%                                09653
      msrecord(msyntax, 0, 0);                                          09654
    % Do special things in help. %                                     09655
      IF inhelp THEN                                                    09656
        BEGIN                                                           09657
        *helpstr* _ "The question mark message for Help is
        missing.  Call ARC!";                                           09658
        IF hlpfileno THEN                                               09659
          BEGIN                                                         09660
          lookadr _                                                     09661
            IF nlmode = fulldisplay THEN $"dqmark" ELSE
            $"tqmark";                                                  09662
          ptr _ orgstid;                                                09663
          ptr.stfile _ hlpfileno;                                       09664
          ptr[1] _ 1;                                                   09665
          lookup($ptr, lookadr, nametyp);                               09666
          IF ptr # endfil THEN                                          09667
            BEGIN                                                       09668
            % Remove name, first EOL. %                                 09669
            IF FIND SF(ptr) [EOL] ^t1 THEN                              09670
              cnveoltcrlf($t1, $helpstr); %copy the rest,
              changing EOL's to CRLF's%;                                09671
            END;                                                        09672
          END;                                                          09673
        fbctl( typecalit, $helpstr);                                    09674
        RETURN;                                                         09675
        END;                                                            09676
    % initialize %                                                     09677
      svrec1 _ recogmode;                                               09678
      svrec2 _ recog2mode;                                              09679
      svllcmd _ FALSE;                                                  09680
      IF nlmode = fulldisplay THEN cmdctl _ 1                           09681
      ELSE cmdctl _ 2;                                                  09682
```

```
      % save current node pointer in path stack %                    09683
       savecnptr _ list.curnodeptr;                                  09684
      % cleanup on any errors %                                      09685
       ON SIGNAL ELSE                                                09686
          BEGIN                                                      09687
          ON SIGNAL ELSE;                                            09688
          IF svl1cmd AND svrec1 = mexpert THEN                       09689
             BEGIN                                                   09690
             recogmode _ svrec1;                                     09691
             recog2mode _ svrec2;                                    09692
             END;                                                    09693
          savecnptr _ list.curnodeptr;                               09694
          END;                                                       09695
      % set up output display area %                                09696
       fbctl( typenulllit );                                         09697
      % build a stack of nodes from the pathh stack %               09698
       CASE prspathstk( cmode ) OF                                   09699
          = 0:                                                       09700
             BEGIN                                                   09701
             &cinstptr _ [pathstk.begnodeptr].addr;                  09702
             WHILE &cinstptr DO                                      09703
                BEGIN                                                09704
                ckshwcmd( &cinstptr );                               09705
                &cinstptr _ cinstptr.alternative;                    09706
                END;                                                 09707
             &cinstptr _ [[pathstk.begnodeptr].alternative].addr;
                                                                     09708
             WHILE &cinstptr DO                                      09709
                BEGIN                                                09710
                ckshwcmd( &cinstptr );                               09711
                &cinstptr _ cinstptr.alternative;                    09712
                END;                                                 09713
             END;                                                    09714
          = 1: ckshwcmd( csstk[0] );                                 09715
          ENDCASE                                                    09716
             BEGIN                                                   09717
             % diddle around to get all possibilites %               09718
                svl1cmd _ cmode.l1cmd := FALSE;                      09719
                IF svl1cmd AND recogmode = mexpert THEN              09720
                   recogmode _ recog2mode _ manticipatory;           09721
             % set up some state info (change later) %               09722
                % *** note:  we are using a new work area to
                control the sequencing through the keyword list,
                but we still use the fbstr from the work area
                presently in effect %                                09723
                   nextchar _ -1; % so nextkey will give back next
                   thing %                                           09724
                   keyinit( &list, $work, cmode, TRUE);              09725
             % loop with nextkey to get all paths %                  09726
                LOOP                                                 09727
                   BEGIN                                             09728
                   IF NOT nextkey( $work, nextchar) THEN EXIT LOOP;
                                                                     09729
                   csstk[csstkx - 1] _ [work.currpath].curnodeptr;
                                                                     09730
                   csstkb _ 0;                                       09731
```

```
                    ckshwcmd( csstk[0] );                                   09732
                    END;                                                    09733
                IF svl1cmd AND svrec1 = mexpert THEN                        09734
                    BEGIN                                                   09735
                    recogmode _ svrec1;                                     09736
                    recog2mode _ svrec2;                                    09737
                    END;                                                    09738
                END;                                                        09739
        % output the helpstring %                                           09740
            inptrf _ 0;                                                     09741
            IF nlmode = fulldispalay THEN fbctl( addcalit )                 09742
            ELSE                                                            09743
                BEGIN                                                       09744
                *helpstr* _ CR, LF, "---";                                  09745
                IF fstrm THEN *helpstr* _ *helpstr*, *fbstr*;               09746
                fbctl( fbendlit, $helpstr);                                 09747
                END;                                                        09748
        % restore current node pointer in pathst %                          09749
            list.curnodeptr _ savecnptr;                                    09750
        RETURN;                                                             09751
        END.
                                                                            09752
    (prspathstk) PROCEDURE                                                  07006
        % FORMALS %                                                         07007
            ( cmode );  % current recognition mode %                        07008
        % LOCALS %                                                          07009
            LOCAL j, jtop, pathptr;                                         07010
        REF pathptr;                                                        07011
                                                                            07012
        csstkx _ csstkb _ 0;                                                07013
        jtop _ pathx / $totalrecsize;                                       07014
        FOR j _ 0 UP UNTIL = jtop DO                                        07015
            BEGIN                                                           07016
            &pathptr _ $pathstk + (j*$totalrecsize);                        07017
            CASE j OF                                                       07018
                = 0:                                                        07019
                    csstk[csstkx := csstkx+1] _ pathptr.curnodeptr;         07020
                ENDCASE                                                     07021
                    CASE pathptr.begnodeptr OF                              07022
                        = pathptr.curnodeptr:                               07023
                            csstk[csstkx := csstkx+1] _ pathptr.curnodeptr;
                                                                            07024
                        ENDCASE fndwhere( &pathptr );                       07025
            END;                                                            07026
        CASE csstkx OF                                                      07027
            = 1:                                                            07028
                IF pathstk.begnodeptr = pathstk.curnodeptr THEN             07029
                    BEGIN                                                   07030
                    csstkx _ -1;                                            07031
                    RETURN( 0 );                                            07032
                    END                                                     07033
                ELSE REPEAT CASE( 2 );                                      07034
            ENDCASE                                                         07035
                CASE [csstk[csstkx-1]].opcode OF                            07036
                    = $keyop:                                               07037
                        CASE kwrstate OF                                    07038
```

```
                    < 0:  RETURN( 1 );                                  07039
                    = 0:                                                07040
                       BEGIN                                            07041
                       BUMP DOWN csstkx;                                07042
                       RETURN( 1 );                                     07043
                       END;                                             07044
                    > 0:  RETURN( 2 );                                  07045
                    ENDCASE;                                            07046
                ENDCASE                                                 07047
                   BEGIN                                                07048
                   CASE [csstk[csstkx-1]].opcode OF                     07049
                      = $option, = $pfcall: BUMP DOWN csstkx;           07050
                      ENDCASE;                                          07051
                   RETURN( 1 );                                         07052
                   END;                                                 07053
END.
                                                                        07054
%%                                                                     07055
```

```
(fndwhere) PROCEDURE                                                    07056
   % FORMALS %                                                          07057
      (pathptr);                                                        07058
   % LOCALS %                                                          07059
      LOCAL instptr;                                                    07060
   REF pathptr, instptr;                                                07061
                                                                        07062
   &instptr _ pathptr.begnodeptr;                                       07063
   WHILE &instptr DO                                                    07064
      IF &instptr = pathptr.curnodeptr THEN                             07065
         BEGIN                                                          07066
         csstk[csstkx := csstkx+1 ] _ pathptr.curnodeptr;               07067
         RETURN;                                                        07068
         END                                                            07069
      ELSE CASE instptr.opcode OF                                       07070
         = $xecute, = $option:                                          07071
            IF fndxec( &instptr, pathptr.curnodeptr ) THEN RETURN
                                                                        07072
            ELSE REPEAT CASE( -1 );                                     07073
         ENDCASE                                                        07074
            IF &instptr = instptr.alternative THEN &instptr _ 0
                                                                        07075
            ELSE &instptr _ instptr.alternative;                        07076
   RETURN;                                                              07077
   END.
                                                                        07078
   %%                                                                  07079
```

```
(fndxec) PROCEDURE                                                    07080
   % FORMALS %                                                        07081
      (xecptr,                                                        07082
      objptr);                                                        07083
   % LOCALS %                                                        07084
      LOCAL instptr;                                                  07085
   REF instptr, xecptr;                                               07086
                                                                      07087
   &instptr _ xecptr.addr;                                            07088
   WHILE &instptr DO                                                  07089
      IF &instptr = objptr THEN                                       07090
         BEGIN                                                        07091
         csstk[csstkx := csstkx+1] _ &xecptr;                         07092
         csstk[csstkx := csstkx+1] _ objptr;                          07093
         RETURN( TRUE );                                              07094
         END                                                          07095
      ELSE CASE instptr.opcode OF                                     07096
         = $xecute, = $option:                                        07097
            BEGIN                                                     07098
            csstk[csstkx := csstkx+1] _ &xecptr;                      07099
            IF fndxec( &instptr, objptr) THEN RETURN( TRUE )          07100
            ELSE BUMP DOWN csstkx;                                    07101
            REPEAT CASE( -1 );                                        07102
            END;                                                      07103
         ENDCASE                                                      07104
            IF &instptr = instptr.alternative THEN &instptr _ 0
                                                                      07105
            ELSE &instptr _ instptr.alternative;                      07106
   RETURN( FALSE );                                                   07107
   END.
                                                                      07108
   %%                                                                07109
```

FINISH

02625

PDATA

```
< NLS, PDATA.NLS;3, >, 20-APR-76 23:45 DSM ;;;;
FILE pdata   %  L10   <REL-NLS>PDATA   %% (L10,) (rel-nls,PDATA.rel,) %
                                                                      02
% CONSTANTS %                                                        0154
   % SIZE DECLARATIONS %                                             0155
      SET EXTERNAL                                                   0156
         prsvsize= 30,  % room for 10 parse rule replacements %      0212
         funrecsize= 10,         % size of funstaterec %            0157
         pathrecsize= 3,         % size of pathsr %                 0158
         totalrecsize= 13,       % total rec size %                 0159
         psdepth= 180,  % total  entries in path stack%             0160
         pssize= 2340,  % psdepth * totalrecsize %                  0161
         selstatesize= 3,        % size of selstate record %        0162
         ptrssize= 20,  % size of ptrstk %                         0163
            % CHANGE DECLARATIONS IN PARSER ALSO %                  0213
               % use following pattern to find spots to change     0215
                  ["ptrssize (in PDATA)"];   %                     0216
         evalsize= 40,  % size of eval stack %                     0164
         sbstksize= 20, % size of sbstack %                        0165
         sbentsize= 2,  % size of sbentry record %                 0166
         framesize= 4,  % size of subsystem definition frame %     0167
         sdptsize= 84,  % size of nlssubs and allsubs %            0168
         fbstrmax= 50,  % max size of feedback string %            0169
         wasize= 15;             % work area word size (5+fbstrmax/5) %
                                                                    0170
            % CHANGE DECLARATIONS IN PARSER ALSO %                  0218
               % use following pattern to find spots to change     0219
                  ["wasize (in PDATA)"];   %                       0220
   % VALIDATION CODE FOR SUBSYSTEM DISPATCH RECORD %               0172
      % This 18 bit validation code is output by the CML compiler and
      checked by dfnsubsys routine to make sure that the pointer being
      handed that routine actually points to a subsystem dispatch record
      %                                                            0173
      SET EXTERNAL                                                 0174
         dptvldationcode= 110473;                                  0227
   % KEYWORD DELIMITER CHARS %                                     0175
      SET EXTERNAL                                                 0176
         eschar= 140B,  % ESC for recognition purposes %           0177
         leftdelim=SP;  % left delimiter for keywords %            0178
   % INTERPRETER EVALUATION MODES %                                0179
      SET EXTERNAL                                                 0180
         unknown= 0,             % dont know what type it is %     0181
         parallel= 1,   % parallel recognition %                   0182
         serial= 2,              % serial recognition %            0183
         parsefunction= 3;       % interpretted by parsing function %
                                                                    0184
   % INTERPRETER OPCODE DECLARATIONS %                             0185
      SET EXTERNAL                                                 0186
         % RECOGNIZERS %                                           0187
            keyop=1B,            % keyword recognition operator %  0188
            confirm=2B, % get command confirmation %                0189
            ssel=3B,             % source selection %              0190
            dsel=4B,             % destination selection %         0191
            lsel=5B,             % literal selection %             0192
            pca=6B,              % ca recognizer %                 0193
            vwspecs=7B, % gets viewspecs %                         0194
```

```
        levadj=10B, % get level adjust string %                          0195
      % OPTIONAL ELEMENTS %                                              0196
        option=11B, % optional parameter %                               0197
        anyof=12B,  % repeated list with failure %                       0198
      % CONTROL ELEMENTS %                                               0199
        pfcall=21B, % parse function call %                              0200
        xecute=22B, % transfer to another point in tree %                0201
        call=23B,           % subroutine call %                           0202
      % FEEDBACK ELEMENTS %                                              0203
        fbclear=31B,        % clear feedback buffer %                     0204
        necho=32B,  % echo noise word string %                           0205
        recho=33B,  % replace last thing echoed %                        0206
      % VALUE MANIPULATIONS %                                            0207
        store=41B,  % store value into variable %                        0208
        pload=42B,          % load variable to ptr stack %                0209
        enter=43B,  % enter constant into stack %                        0210
        valueof=44B;        % valueof built in function %                 0211
% parser flags %                                                          03
  DECLARE EXTERNAL                                                       04
  litTakeDown= 1;       % TRUE IFF literal is to be taken down (see
  <select,getlit> %                                                      05
% repeat command state info %                                           06
  DECLARE EXTERNAL STRING                                                07
    keyinpstr[50];    % actual chars typed to get recognition %          08
  DECLARE EXTERNAL STRING                                                09
    keyrptstr[50];    % recognition string at end of last command %
                                                                         010
  DECLARE EXTERNAL                                                       011
    keysaveflag= 1;   % TRUE if keyword inpt chars are to be saved %
                                                                         012
% parser variables %                                                    013
  DECLARE EXTERNAL                                                       014
    %directory and archive directory commands%                          015
      gparam,                                                            016
      gparm2,                                                            017
      gparm3,                                                            018
      gparm4,                                                            019
    aheadfilename,                                                       020
    dent,                                                                021
    dest,                                                                022
    ent,                                                                 023
    filtre,                                                              024
    namfil,                                                              025
    ff,                                                                  026
    fromwhom,                                                            027
    level,                                                               028
    literal,                                                             029
    param,                                                               030
    param1,                                                              031
    param2,                                                              032
    param3,                                                              033
    param4,                                                             0149
    pb,                                                                  034
    port,                                                                035
    retfilename,                                                         036
    s1,                                                                  037
```

```
      s2,                                                              038
      sent,                                                            039
      sim,                                                             040
      source,                                                          041
      vs;                                                              043
%These symbols mysteriously disappeared from PDATAin late 1974%        0231
   DECLARE EXTERNAL                                                    0232
      curmkr[2],                                                       0233
      cspupdate,                                                       0234
      usesrr,                                                          0235
      cspcacode,                                                       0236
      cspusqcod,                                                       0237
      cspvs[2];                                                        0238
% writeable data for parser %                                         044
   % INTERPRETER CONTROL VARIABLES %                                  045
      % PTR TO THE INPUT ROUTINE %                                    063
         DECLARE EXTERNAL                                            064
            inpt;                                                    065
      % CURRENT GRAMMAR %                                            066
         DECLARE EXTERNAL                                            067
            grammar;    % ptr to start of grammar %                   068
      % FRAME COUNTER %                                              069
         DECLARE EXTERNAL                                            070
            framecounter= 0;                                         071
      % CUT BACK POINTER %                                           0224
         DECLARE EXTERNAL                                            0225
            cutstop= 0;                                              0226
      % TERMINATION CHARACTERS FOR NLS COMMANDS %                    072
         DECLARE EXTERNAL                                            073
            cachar=4,           % ^D %                              074
            optchar=25B,        % ^U %                              075
            rptchar=140B,       % ^B %                              076
            inschar=5;          % ^E %                              077
   % INTERPTETER STATE DATA %                                        078
      % PROMPTS STRING %                                            079
         DECLARE EXTERNAL STRING                                     080
            prompstr[72];                                            081
      % COMMAND HEARALD STRING %                                    082
         DECLARE EXTERNAL STRING                                     083
            hrldstr[30];        % TNLS hearald string %              0230
      % STRING FOR PROMPTS FOR PFCALL FUNCTIONS %                    084
         DECLARE EXTERNAL STRING                                     085
            savhelpstr[25];                                          0229
      % SUBSYSTEM NAME STRING %                                      086
         DECLARE EXTERNAL STRING                                     087
            ssysname[30];                                            0228
      % SUBSYSTEM STACK %                                           088
         % The subsystem stack (sbstack) contains a one word entry for
         each subsystem executed.  The information in the stack drives
         the supervisor control routine.  The execution functions for
         the GOTO, QUIT, and EXECUTE commands manipulate entries in this
         stack %                                                     089
         DECLARE EXTERNAL                                            090
            sbstkx=0,   % index to next entry in sbstack %           091
            sbstack[ sbstksize ];       % subsystem control stack %  092
      % SUBSYSTEM DISPATCH STACK %                                   093
```

```
   DECLARE EXTERNAL                                                    094
      sdptx= 0,   % index to next cell in nlssubs %                    095
      nlssubs[ sdptsize ], % attached subsystems dispatch stack %
                                                                       096
      sdptxa= 0,  % index to next cell in allsubs %                    0222
      allsubs[ sdptsize ]; % all subsystems dispatch stack %           0223
% RECOGNITION STATE VARIABLES %                                       097
   DECLARE EXTERNAL                                                    098
      cmdmode,    % command recognition flag %                        099
      nofail=1,   % TRUE if recognizers cannot fail %                 0100
      fbstr;      % ptr to current feedback string %                 0101
         % this pointer is kept in a global to avoid having to
         pass it around as a actual argument among the feedback
         utility routines.  Note that the actual feedback string
         resides in the stack frame.  fbptr must be reconstructed
         when popping states so it reflects the "current" feedback
         string %                                                      0102
% REPLACE PARSE RULE SAVE AREA %                                     0103
   DECLARE EXTERNAL                                                    0104
      prsavx= 0,  % index to next cell in prsavearea %                 0105
      prsavearea[prsvsize];       % save area for replaced
      interpreter instructions %                                       0106
% COMMAND COMPLETION CODE %                                          0107
   DECLARE EXTERNAL                                                    0108
      complcode= 1;                                                    0109
% ACTION DATA BLOCKS %                                               0110
   DECLARE EXTERNAL                                                    0111
      defaction[2],% record for processing default chars %             0112
      caaction[2],        % record for processing ca char %           0113
      optaction[2];       % record for processing opt char %          0114
% FLAGS %                                                            0115
   DECLARE EXTERNAL                                                    0116
      slink= 0,   % TRUE when doing a link/filelink selection %
                                                                       0150
      nwlnk= 0,   % TRUE when doing a new filelink selection %
                                                                       0153
      cdlnk= 0,   % TRUE to put comma after dir name  %               0151
      clpsw= 0,   % TRUE means doing a password selection %           0152
      auxinflag= 0,       % TRUE when reading input from aux
      source %                                                         0117
      basestateflag= 0,   % TRUE when interpreter is in base state
      %                                                                0118
      needconfirm=0, % flag for confirms after bugs %                 0119
      lastsel;% previous selection opcode %                            0120
   DECLARE EXTERNAL                                                    0121
      cueflg;     % TRUE when inprompts have been done %             0122
% PATH STACK %                                                       0123
   % the path stack contains entries each of which are composed of
   two records.  The first record is the control record for the
   path stack entry and is used only by the interpreter to record
   interpreter state information for the entry.  The second record
   is the function state record and it is used by the processing
   function to record return values and save any local state
   associated with the function %                                    0124
   % If a processing function requires more space for recording
   state information than is available in the function state
```

```
      record, then it should allocate the needed space and record a
      pointer to the allocated space in the function state record.
      Note that the function has the responsibility for recording its
      own state information and allocating and deallocating space as
      required.  The interpreter passes control to the functions,
      passing them a pointer to the function state record and a
      control word, and it is up the the function to manage the
      saving of state if required. %                                  0125
                                                                      0126
      DECLARE EXTERNAL                                                0127
         pathx=0,    % next available cell in path stack %             0128
         pathstk[pssize],% storage for path stack entries %           0129
         pathbase= pathstk;  % ptr to base for this subsys %          0130
    % EXECUTION POINTER STACK %                                       0131
      % the execution pointer stack (ptrstk) whose next position is
      indexed by ptrx contains pointers to "xecute" instructions in
      the grammar and marks the points where control has been
      transfered out of line.  This stack is external because it is
      also manipulated by the keyword recognition processor, which
      must chase down the alternate control paths in order to
      discover the actual path %                                      0132
      DECLARE EXTERNAL                                                0133
         ptrx= 0,    % index of next position in ptrstk %             0134
         ptrstk[ ptrssize ]; % stack for recording tree switches %
                                                                      0135
    % EVAL STACK %                                                   0136
      DECLARE EXTERNAL                                                0137
         evalx=0,    % next available cell in eval stack %           0138
         evalstk[evalsize];% storage for eval stack %                 0139
% HELP SUBSYSTEM INTERFACE POINTERS %                                 0140
   DECLARE EXTERNAL                                                   0141
      hlpcmdstk= 0,     % ptr to array with command path to be shown to
      user; 0 if array not allocated %                                0142
      hpcmdmax= 50;     % maximum number of words in path %           0143
% jump return rings data%                                           0144
   DECLARE EXTERNAL                                                   0145
      trrcnt,  %used for jump file return%                            0146
      srrcnt; %used for jump return%                                  0147
FINISH of pdata                                                       0148
```

PRM SPC

```
< NLS, PRMSPC.NLS;4, >, 19-OCT-76 17:07 DSM ;;;;
FILE prmspc  %  L10  <REL-NLS>PRMSPC  %% (L10,) (rel-nls,PRMSPC.rel,) %
                                                                      02
%variable declarations%                                               04
   REF rawchr, tda;                                                 01959
   DECLARE                                                            06
      name=1, word=2, contnt=3, sid=4, gotchr=0, getchr=1,            07
      frstchr=1;                                                      08
   REGISTER a1 = 12, m = 10;
                                                                    01299
%INPUT support routines%
                                                                     0880
   (inbug) PROCEDURE(bg);                                            0881
      an();                                                         01055
      arm();                                                        01056
      CASE lookc() OF                                                0883
         =CD:                                                        0884
            BEGIN                                                    0972
            input();                                                 0973
            GOTO STATE;                                              0971
            END;                                                     0974
         =CA, =C.:                                                   0885
            BEGIN                                                    0886
            input();                                                 0975
            strbug(bg);                                              0887
            RETURN ;                                                 0888
            END;                                                     0889
         =BC:                                                        0890
            BEGIN                                                    0891
            input();                                                 0976
            BUMP [m];                                                0892
            RETURN ;                                                 0893
            END;                                                     0894
         ENDCASE                                                     0895
            BEGIN                                                    0896
            BUMP [m], [m];                                           0898
            RETURN;                                                  0899
            END;                                                     0900
      END.
                                                                     0901
   (instid) PROCEDURE(bg);                                         01126
      LOCAL cnt;                                                   01153
      an();                                                        01127
      arm();                                                       01128
      CASE lookc() OF                                              01129
         =CD:                                                      01130
            BEGIN                                                  01131
            input();                                               01132
            GOTO STATE;                                            01133
            END;                                                   01134
         =BUG, =C.:                                                01135
            BEGIN                                                  01136
            input();                                               01137
            strbug(bg);                                            01138
            RETURN ;                                               01139
            END;                                                   01140
```

```
      =BC:                                                          01141
         BEGIN                                                      01142
         input();                                                   01143
         BUMP [m];                                                  01144
         RETURN ;                                                   01145
         END;                                                       01146
      =SP:                                                          01154
         BEGIN                                                      01155
         input();                                                   01156
         *stn* _ NULL;                                              01157
         cnt _ 0;                                                   01158
         [bg] _ origin;                                             01234
         [bg].stfile _ lcfile();                                    01235
         inname($stn);  %can return +1, +2, +3 past call%           01159
         BUMP cnt, cnt;                                             01160
         CASE cnt OF                                                01161
            =2:                                                     01162
               BEGIN                                                01163
               input();                                             01378
               specreg($stn, name, bg);                             01164
               IF [bg] = endfil THEN                                01294
                  err($"No such statement encountered");            01295
               RETURN;                                              01165
               END;                                                 01166
            =1:                                                     01167
               BEGIN                                                01168
               BUMP [m];                                            01169
               RETURN;                                              01170
               END;                                                 01171
            ENDCASE                                                 01172
               BEGIN                                                01173
               BUMP [m],[m];                                        01174
               RETURN;                                              01175
               END;                                                 01176
         END;                                                       01177
      ENDCASE                                                       01147
         BEGIN                                                      01148
         BUMP [m], [m];                                             01149
         RETURN;                                                    01150
         END;                                                       01151
   END.
                                                                    01152
(intext)                                                             0272
   %This routine reads literal input from a display terminal, appends
   it to an A-string, and displays it in the literal feedback area.
   The argument passed this routine should be the address of the
   A-string to which the literal input is to be appended. (Note that
   this routine does not clear the A-string before it begins reading
   characters.)  TEXT = arbitrary number of characters up  to but not
   including a CA or Center dot. DOES A SKIP RETURN IF A BC OR BW IS
   INPUT AND THE STRING IS EMPTY%                                    0273
   %---------------%                                                 0279
                                                                    02080
   PROCEDURE(astrng);                                                0280
   REF astrng;                                                       0282
   &tda _ lda(); %for tabs%                                         02063
```

```
    af();                                                              0283
    disarm();                                                         01178
    setlit();  %set up for literal collection%                        02081
    CASE rdlit(&astrng, 0) OF                                         02087
       = 1: %CA or CDOT%                                              02088
          BEGIN                                                       02090
          unput();                                                    01962
          RETURN;                                                      0295
          END;                                                        02091
       = 3: %BC or BW%                                                02089
          BEGIN                                                       02092
          BUMP [m];                                                    0987
          RETURN;                                                      0988
          END;                                                        02093
       ENDCASE RETURN;                                                02094
    END.                                                               0306
                                                                       0307
(innum)                                                                0308
    %This routine reads digits from the work station, appends          0309
    them to an A-string, and displays them in the name register        0310
     The argument is the address of the register                       0313
    into which the A-string is to be put. (Note that this              0314
    routine does not clear the A-string before appending               0315
    characters to it.).%                                               0316
    %---------------%                                                 0317
    PROCEDURE(astrng);                                                 0318
    LOCAL char;                                                        0319
    REF astrng;                                                        0320
    af();                                                              0321
    disarm();                                                         01179
    LOOP                                                               0322
       BEGIN                                                           0323
       dn(&astrng);                                                    0324
       CASE char _ lookc() OF                                          0325
          = '-:                                                       01180
             IF astrng.L = empty THEN                                 01182
                *astrng* _ *astrng*, char                             01183
             ELSE                                                     01184
                BEGIN                                                 01185
                BUMP [m], [m];                                        01186
                RETURN;                                               01187
                END;                                                  01188
          =CA, =C.:                                                    0326
             BEGIN                                                     0327
             IF astrng.L = empty THEN                                 01189
                BEGIN                                                 01190
                input();                                              01191
                strbug($b1);                                          01192
                <TXTEDT, ndr>($b1, $p1, $p2);                         01193
                *astrng* _ p1 p2;                                     01194
                END;                                                  01195
             RETURN;                                                   0329
             END;                                                      0330
          =CD:                                                         0331
             BEGIN                                                     0332
             input();                                                  0999
```

```
            an();                                                    0333
            GOTO STATE;                                              0334
            END;                                                     0335
         =D:                                                         0336
            IF astrng.L >= astrng.M THEN                            01463
               dismes(2, $"number too long -- last character(s)
               lost")                                               01464
            ELSE *astrng* _ *astrng*, char;                         01465
         =BC:                                                        0337
            IF astrng.L = empty THEN                                01001
               BEGIN                                                01002
               input();                                             01003
               BUMP [m];                                            01004
               RETURN;                                              01005
               END                                                  01006
            ELSE <INPFBK, bkc>(&astrng);                            01007
         =BW:                                                        0338
            IF astrng.L = empty THEN                                01008
               BEGIN                                                01009
               input();                                             01010
               BUMP [m];                                            01011
               RETURN;                                              01012
               END                                                  01013
            ELSE <INPFBK, bkw>(&astrng);                            01014
         ENDCASE                                                     0339
            BEGIN                                                    0340
            BUMP [m], [m];                                          01015
            RETURN;                                                 01016
            END;                                                     0343
      input();                                                      01017
      END;                                                           0344
   END.                                                              0345
                                                                     0346
(insr)                                                               0347
   %This routine reads characters from the work station,             0348
   appends them to an A-string, and displays them in the name        0349
   register.  The argument should be the address of the              0350
   register into which the A-string is to be put. (Note that         0351
   this routine does not clear the A-string before appending         0352
   characters to it.)%                                               0353
   %---------------%                                                0354
   PROCEDURE(astrng);                                                0355
   LOCAL char;                                                       0356
   REF astrng;                                                       0357
   af();                                                             0358
   disarm();                                                        01196
   LOOP                                                              0359
      BEGIN                                                          0360
      dn(&astrng);                                                   0361
      CASE char _ lookc() OF                                         0362
         =CD:                                                        0363
            BEGIN                                                    0364
            input();                                                01018
            GOTO STATE;                                              0366
            END;                                                     0367
         =CA:                                                        0368
```

```
            BEGIN                                                   01380
            IF astrng.L = empty THEN incont(&astrng);               01381
            RETURN;                                                 01382
            END;                                                    01383
         =C.:                                                       01379
            RETURN;                                                  0371
         =BC:                                                       01019
            IF astrng.L = empty THEN                                01020
               BEGIN                                                01021
               input();                                             01022
               BUMP [m];                                            01023
               RETURN;                                              01024
               END                                                  01025
            ELSE <INPFBK, bkc>(&astrng);                            01026
         =BW:                                                       01027
            IF astrng.L = empty THEN                                01028
               BEGIN                                                01029
               input();                                             01030
               BUMP [m];                                            01031
               RETURN;                                              01032
               END                                                  01033
            ELSE <INPFBK, bkw>(&astrng);                            01034
         ENDCASE                                                     0375
            IF astrng.L >= astrng.M THEN                            01467
               dismes(2, $"name too long -- last character(s) lost")
                                                                    01468
            ELSE                                                    01469
               *astrng* _ *astrng*, char;                           01466
      input();                                                      01035
      END;                                                           0376
   END.                                                              0377
                                                                     0378
(incont) PROCEDURE(string);                                         01282
   %puts bugged text entity in string passed it%                    01283
   LOCAL TEXT POINTER bug1, bug2, ptr1, ptr2;                       01284
   REF string;                                                      01285
   INPUT BUG bug1 BUG bug2;                                         01286
   <TXTEDT, tdr> ($bug1, $bug2, $ptr1, $ptr2);                      01287
   *string* _ ptr1 ptr2;                                            01288
   dn(&string);                                                     01333
   RETURN;                                                          01289
   END.


                                                                    01290
(invsbl) PROCEDURE(astrng);                                         01202
   CASE innmwd(astrng, FALSE, $vdr) OF                              01203
      =0:                                                           01204
         RETURN;                                                    01205
      =1:                                                           01206
         BEGIN                                                      01207
         BUMP [m];                                                  01208
         RETURN;                                                    01209
         END;                                                       01210
      ENDCASE                                                       01211
         BEGIN                                                      01212
         BUMP [m],[m];                                              01213
```

```
        RETURN;                                                    01214
        END;                                                       01215
    END.
                                                                   01216
(inword) PROCEDURE(astrng);                                        01349
  CASE innmwd(astrng, TRUE, $nmdr) OF                              01350
    =0:                                                            01351
      RETURN;                                                      01352
    =1:                                                            01353
      BEGIN                                                        01354
      BUMP [m];                                                    01355
      RETURN;                                                      01356
      END;                                                         01357
    ENDCASE                                                        01358
      BEGIN                                                        01359
      BUMP [m],[m];                                                01360
      RETURN;                                                      01361
      END;                                                         01362
    END.
                                                                   01363
(inname) PROCEDURE(astrng);                                        01217
  CASE innmwd(astrng, FALSE, $nmdr) OF                             01218
    =0:                                                            01219
      RETURN;                                                      01220
    =1:                                                            01221
      BEGIN                                                        01222
      BUMP [m];                                                    01223
      RETURN;                                                      01224
      END;                                                         01225
    ENDCASE                                                        01226
      BEGIN                                                        01227
      BUMP [m],[m];                                                01228
      RETURN;                                                      01229
      END;                                                         01230
    END.
                                                                   01231
(innmwd)                                                           0379
  %This routine reads characters from the work station, appends them
  to a register, and displays them in the nameregister.  Alphabetic
  characters are forced to upper case before insertion into the
  A-string if the wordflag is false.  The argument should be the
  address of the register into which the A-string is to be put.
  (Note that this routine does not clear the A-string before
  appending characters to it.).%                                   0380
  %---------------%                                               0387
  PROCEDURE(astrng, wordflag, delimproc);                          0388
  LOCAL char;                                                      0389
  REF astrng, delimproc;                                           0390
  af();                                                            0391
  disarm();                                                       01197
  LOOP                                                             0392
    BEGIN                                                          0393
    dn(&astrng);                                                   0394
    CASE char _ lookc() OF                                         0395
      =CD:                                                         0396
        BEGIN                                                      0397
```

```
            input();                                                    01036
            GOTO STATE;                                                  0399
            END;                                                         0400
         =CA, =C.:                                                       0401
            BEGIN                                                       01198
            IF astrng.L = empty THEN                                     0404
               BEGIN                                                     0405
               input();                                                 01292
               strbug($b1);                                             01037
               delimproc($b1, $p1, $p2);                                0407
               IF wordflag THEN *astrng* _ p1 p2                        0408
               ELSE *astrng* _ +p1 p2;                                  01232
               dn(&astrng);                                             0409
               END;                                                     0411
            RETURN(0);                                                  0412
            END;                                                        0413
         =BC:                                                           01038
            IF astrng.L = empty THEN                                    01039
               BEGIN                                                    01040
               input();                                                 01041
               RETURN(1);                                               01043
               END                                                      01044
            ELSE <INPFBK, bkc>(&astrng);                                01045
         =BW:                                                           01046
            IF astrng.L = empty THEN                                    01047
               BEGIN                                                    01048
               input();                                                 01049
               RETURN(1);                                               01051
               END                                                      01052
            ELSE <INPFBK, bkw>(&astrng);                                01053
         ENDCASE                                                        0416
            BEGIN                                                       0417
            IF NOT wordflag AND char IN ['a,'z] THEN                    0418
               char _ char - 40B;                                       01233
            IF astrng.L >= astrng.M THEN                                01470
               dismes(2, $"name too long -- last character(s) lost")
                                                                        01471
            ELSE                                                        01472
               *astrng* _ *astrng*, char;                               0419
            END;                                                        0420
      input();                                                          01054
      END;                                                              0421
   END.                                                                 0422
                                                                        0423
(inch) PROCEDURE(char);                                                 01102
   LOCAL nchar;                                                         01371
   af();                                                                01199
   disarm();                                                            01200
   CASE nchar _ lookc() OF                                              01103
      =char:                                                            01104
         BEGIN                                                          01105
         input();                                                       01106
         RETURN;                                                        01107
         END;                                                           01108
      =CD:                                                              01109
         BEGIN                                                          01110
```

```
         input();                                                    01111
         GOTO STATE;                                                 01112
         END;                                                        01113
      =BC:                                                           01114
         BEGIN                                                       01115
         input();                                                    01116
         BUMP [m];                                                   01117
         RETURN ;                                                    01118
         END;                                                        01119
      ENDCASE                                                        01120
         BEGIN                                                       01121
         BUMP [m],[m];                                               01122
         RETURN;                                                     01123
         END;                                                        01124
   END.
                                                                     01125
(oldanswer) %this procedure accepts a yes or no answer from the
keyboard, and returns with a 0 if the answer was negative, and a 1 if
it was positive%                                                     01424
   %------------------%                                             01425
   PROCEDURE;                                                        01426
   CASE nlmode OF                                                    02167
      = fulldisplay:                                                 02168
         CASE inpcuc() OF                                            01427
            = 'Y, = CA, = SP:                                        01428
               BEGIN                                                 01429
               DSP(?OFF Yes);                                        01430
               RETURN(TRUE);                                         01431
               END;                                                  01432
            = 'N:                                                    01433
               BEGIN                                                 01434
               DSP(?OFF No);                                         01435
               RETURN(FALSE)                                         01436
               END;                                                  01437
            = CD: GOTO STATE;                                        01438
            ENDCASE                                                  01439
               BEGIN                                                 01440
               qm();                                                 01441
               REPEAT;                                               01442
               END;                                                  01443
      = typewriter:                                                  02169
         BEGIN                                                       02200
         echoff();                                                   02174
         CASE inpcuc() OF                                            02175
            = 'Y, =SP, = CA:                                         02176
               BEGIN                                                 02177
               echo($"Yes ");                                        02178
               curchr _ 'Y;                                          02179
               RETURN(TRUE);                                         02180
               END;                                                  02181
            = 'N:                                                    02182
               BEGIN                                                 02183
               echo($"No ");                                         02184
               curchr _ 'N;                                          02185
               RETURN(FALSE)                                         02186
               END;                                                  02187
```

```
            = CD: GOTO STATE;                                          02188
            = '?:                                                      02189
               BEGIN                                                   02190
               typeas($"
               Type CD to abort; 'y, SP, or CA for YES; 'n for NO:
               ");                                                     02191
               REPEAT;                                                 02192
               END;                                                    02193
            ENDCASE                                                    02194
               BEGIN                                                   02195
               typeas($" ?? ");                                        02196
               REPEAT;                                                 02197
               END;                                                    02198
         END;                                                          02201
      ENDCASE err($"Illegal parsing mode detected in ANSWER");         02170
   END.

                                                                       01444
(infilename) PROCEDURE (string);  %input a file name (DNLS)%           02095
   LOCAL bugf;                                                         02104
   LOCAL TEXT POINTER bug, start, end;                                 02105
   REF string;                                                         02106
   LOOP                                                                02109
      BEGIN                                                            02110
      INPUT (BUG bug bugf _ TRUE; / VISIBLE string bugf _ FALSE;);
                                                                       02096
      IF bugf THEN                                                     02097
         BEGIN                                                         02098
         flndr($bug, $start, $end);                                    02099
         *string* _ start end;                                         02100
         dn(&string);                                                  02108
         END;                                                          02101
      CASE input() OF                                                  02112
         = BC:                                                         02124
            BEGIN                                                      02125
            string.L _ MAX(string.L-1, 0);                             02129
            INPUT VISIBLE string;                                      02128
            REPEAT CASE;                                               02130
            END;                                                       02126
         = CA : RETURN;                                                02119
         = CD : GOTO STATE;                                            02120
         ENDCASE                                                       02121
            BEGIN                                                      02122
            IF bugf THEN delbm();                                      02116
            *string* _ NULL;                                           02117
            dn(&string);                                               02118
            END;                                                       02123
      END;                                                             02111
   END.                                                                02103
(invspc) PROCEDURE (srcdpa);                                            0458
   %This routine reads characters from the work station and changes
   viewspecs.  The original viewspecs are taken from the appropriate
   display area and stored in sysvspec.  The viewspecs changed are
   those in sysvspec.  That's where the new viewspecs will be on
   return..  It sets the viewspecs large upon entry, and small upon
   exit.
   A CD does a GOTO STATE.  A CA or Centerdot causes a return of the
```

```
   display area where the cursor is.  No need to call dpset.%   01451
   %---------------%                                            0465
   LOCAL da, save, sav1, sav2;                                  01642
   REF srcdpa, da;                                              01453
   flagut(20, $setfg);  %to see if this procedure is being used%
                                                                02244
   savvspc (&srcdpa);   %move viewspecs to sysvspec%            0468
   dspvsp (sysvspec, sysvspec[1], 3);   %display them%          01450
   ltl ();                                                      0469
   af ();                                                       0470
   RESET savevspec;                                             01647
   *vspstr* _ NULL;                                             01648
   CASE input () OF                                             0472
      =CD:                                                      0473
         BEGIN                                                  0474
         savvspc (&srcdpa);                                     0475
         lts ();                                                0476
         dn($"");                                               01651
         an ();                                                 0477
         GOTO STATE;                                            0478
         END;                                                   0479
      =CA, =C.:                                                 0480
         BEGIN                                                  0481
         lts ();                                                0483
         an ();                                                 0484
         dn($"");                                               01650
         RETURN (lda ());                                       0485
         END;                                                   0486
      ='f: %recreate the display%                               01625
         BEGIN                                                  01628
         &da _ lda ();                                          01629
         IF sysvspec.vsrlev THEN                                01630
            sysvspec _ <SEQGEN, reslev>(sysvspec, getlev(da.dacsp));
                                                                01631
         save _ da.davspec.vsdaft := TRUE;                      01634
         sav1 _ da.davspec := sysvspec;                         01643
         sav2 _ da.davspc2 := sysvspec[1];                      01644
         dafrmt (&da, 0);                                       01635
         da.davspec.vsdaft _ save;                              01636
         da.davspec _ sav1;                                     01645
         da.davspc2 _ sav2;                                     01646
         REPEAT CASE;                                           01649
         END                                                    01641
      ENDCASE %includes BC%                                     0495
         BEGIN                                                  0496
         IF curchr = BC AND  vspstr.L = empty THEN RETURN(0);   01675
         sysvspec _                                             0497
            stklt (curchr, sysvspec, sysvspec[1] : sysvspec[1]);
                                                                01452
         dn($vspstr);                                           01655
         dspvsp (sysvspec, sysvspec[1], 3);                     0498
         REPEAT CASE;                                           0499
         END;                                                   0500
   END.                                                         0501
                                                                0502
(inievadj) PROCEDURE(astrng);                                   01676
```

```
      %this procedure gets characters for LEVADJ.  It collects
      characters ontil it encounters something other than a U or D (or
      backspace).  It puts these characters into the string passed it
      and displays them in the name register.  Note that the string will
      be empty upon RETURN if the user does no level adjust.%       01677
      %--------------%                                              01678
      LOCAL char;                                                   01679
      REF astrng;                                                   01680
      *astrng* _ NULL;                                              01681
      LOOP                                                          01682
         BEGIN                                                      01683
         dn(&astrng);                                               01684
         CASE (char _ lookc()) OF                                   01685
            =CA, =C.:                                               01717
               BEGIN                                                01719
               EXIT LOOP;                                           01720
               END;                                                 01721
            =SP:                                                    01686
               BEGIN                                                01687
               input();                                             01688
               EXIT LOOP;                                           01689
               END;                                                 01690
            ='u, ='d:                                               01691
               *astrng* _ *astrng*, char;                           01692
            =BC:                                                    01693
               IF astrng.L > empty THEN BUMP DOWN astrng.L          01694
               ELSE                                                 01695
                  BEGIN                                             01696
                  input();                                          01697
                  BUMP [m];                                         01698
                  EXIT LOOP;                                        01699
                  END;                                              01700
            =BW, =$ascbst: *astrng* _ NULL;                         01701
            =CD:                                                    01702
               BEGIN                                                01703
               input();                                             01704
               GOTO STATE;                                          01705
               END;                                                 01706
            ENDCASE BEGIN                                           01707
               reseth( &astrng ); % put collected chars back into input
               buffer %                                             01708
               *astrng* _ NULL;                                     01709
               dn( &astrng );  % clear name buffer %                01710
               EXIT LOOP;                                           01711
               END;                                                 01712
         input();                                                   01713
         END;                                                       01714
      RETURN;                                                       01715
      END.
                                                                    01716
%input BUG's CA%
                                                                    01057
   (inlca) PROC(bg1);                                               01058
      LOCAL cdot;                                                   01059
      REF bg1;                                                      01060
      INPUT BUG bg1 (CA cdot _ 0; / C. cdot _ 1);                   01061
```

```
    RETURN (cdot) END.
                                                                    01062
(in2ca) PROC(bg1, bg2);                                             01063
    LOCAL cdot;                                                     01064
    REF bg1, bg2;                                                   01065
    INPUT BUG bg1 BUG bg2 (CA cdot _ 0; / C. cdot _ 1);             01066
    RETURN (cdot) END.
                                                                    01067
(in3ca) PROC(bg1, bg2, bg3);                                        01068
    LOCAL cdot;                                                     01069
    REF bg1, bg2, bg3;                                              01070
    INPUT BUG bg1 BUG bg2 BUG bg3 (CA cdot _ 0; / C. cdot _ 1);     01071
    RETURN (cdot) END.
                                                                    01072
(in4ca) PROC(bg1, bg2, bg3, bg4);                                   01073
    LOCAL cdot;                                                     01074
    REF bg1, bg2, bg3, bg4;                                         01075
    INPUT BUG bg1 BUG bg2 BUG bg3 BUG bg4                           01076
      (CA cdot _ 0; / C. cdot _ 1);                                 01334
    RETURN (cdot) END.
                                                                    01078
%input BUG's TEXT CA%
                                                                    01079
(in0tca) PROC(astrng);                                              01080
    LOCAL cdot;                                                     01081
    REF astrng;                                                     01082
    INPUT TEXT astrng (CA cdot _ 0; / C. cdot _ 1);                 01083
    RETURN (cdot) END.
                                                                    01084
(in1tca) PROC(bg1, astrng);                                         01085
    LOCAL cdot;                                                     01086
    REF bg1, astrng;                                                01087
    INPUT BUG bg1 TEXT astrng (CA cdot _ 0; / C. cdot _ 1);         01088
    RETURN (cdot) END.
                                                                    01089
(in2tca) PROC(bg1, bg2, astrng);                                    01090
    LOCAL cdot;                                                     01091
    REF bg1, bg2, astrng;                                           01092
    INPUT BUG bg1 BUG bg2 TEXT astrng (CA cdot _ 0; / C. cdot _ 1);
                                                                    01093
    RETURN (cdot) END.
                                                                    01095
(in3tca) PROC(bg1, bg2, bg3, astrng);                               01096
    LOCAL cdot;                                                     01097
    REF bg1, bg2, bg3, astrng;                                      01098
    INPUT BUG bg1 BUG bg2 BUG bg3 TEXT astrng                       01099
      (CA cdot _ 0; / C. cdot _ 1);                                 01100
    RETURN (cdot) END.
                                                                    01101
%input STID's CA%
                                                                     0903
(in1sca) PROC(bg1);                                                  0904
    LOCAL cdot;                                                      0906
    REF bg1;                                                         0905
    INPUT STID bg1 (CA cdot _ 0; / C. cdot _ 1);                     0907
    RETURN (cdot) END.
```

```
                                                                    0908
   (in2sca) PROC(bg1, bg2);                                         0914
      LOCAL cdot;                                                   0915
      REF bg1, bg2;                                                 0916
      INPUT STID bg1 STID bg2 (CA cdot _ 0; / C. cdot _ 1);         0917
      RETURN (cdot) END.
                                                                    0918
   (in3sca) PROC(bg1, bg2, bg3);                                    0909
      LOCAL cdot;                                                   0910
      REF bg1, bg2, bg3;                                            0911
      INPUT STID bg1 STID bg2 STID bg3                              0912
         (CA cdot _ 0; / C. cdot _ 1);                             01373
      RETURN (cdot) END.
                                                                    0913
   (in4sca) PROC(bg1, bg2, bg3, bg4);                               0919
      LOCAL cdot;                                                   0920
      REF bg1, bg2, bg3, bg4;                                       0921
      INPUT STID bg1 STID bg2 STID bg3 STID bg4                     0922
         (CA cdot _ 0; / C. cdot _ 1);                              0924
      RETURN (cdot) END.
                                                                    0923
%input STID's TEXT CA%
                                                                    0925
   (in0stca) PROC(astrng);                                          0926
      LOCAL cdot;                                                   0927
      REF astrng;                                                   0928
      INPUT TEXT astrng (CA cdot _ 0; / C. cdot _ 1);               0929
      RETURN (cdot) END.
                                                                    0930
   (in1stca) PROC(bg1, astrng);                                     0977
      LOCAL cdot;                                                   0978
      REF bg1, astrng;                                              0979
      INPUT STID bg1 TEXT astrng (CA cdot _ 0; / C. cdot _ 1);      0980
      RETURN (cdot) END.
                                                                    0981
   (in2stca) PROC(bg1, bg2, astrng);                                0931
      LOCAL cdot;                                                   0932
      REF bg1, bg2, astrng;                                         0933
      INPUT STID bg1 STID bg2 TEXT astrng                           0934
         (CA cdot _ 0; / C. cdot _ 1);                             01374
      RETURN (cdot) END.
                                                                    0935
   (in3stca) PROC(bg1, bg2, bg3, astrng);                           0936
      LOCAL cdot;                                                   0937
      REF bg1, bg2, bg3, astrng;                                    0938
      INPUT STID bg1 STID bg2 STID bg3 TEXT astrng                  0939
         (CA cdot _ 0; / C. cdot _ 1);                              0948
      RETURN (cdot) END.
                                                                    0940
%input STID's LEVADJ%
                                                                    0949
   (in1sadj) PROC(bg1, astrng);                                     0950
      LOCAL cdot;                                                   0951
      REF bg1, astrng;                                              0952
      *astrng* _ NULL;                                              0953
      INPUT STID bg1 LEVADJ astrng                                  0954
```

```
      (CA cdot _ 0; / C. cdot _ 1);                                    0955
   RETURN (cdot) END.
                                                                       0956
(in2sadj) PROC(bg1, bg2, astrng);                                      0957
   LOCAL cdot;                                                         0958
   REF bg1, bg2, astrng;                                               0959
   *astrng* _ NULL;                                                    0960
   INPUT STID bg1 STID bg2 LEVADJ astrng                               0961
      (CA cdot _ 0; / C. cdot _ 1);                                   01348
   RETURN (cdot) END.
                                                                       0963
(in3sadj) PROC(bg1, bg2, bg3, astrng);                                 0964
   LOCAL cdot;                                                         0965
   REF bg1, bg2, bg3, astrng;                                          0966
   *astrng* _ NULL;                                                    0967
   INPUT STID bg1 STID bg2 STID bg3 LEVADJ astrng                      0968
      (CA cdot _ 0; / C. cdot _ 1);                                    0969
   RETURN (cdot) END.
                                                                       0970
%viewspec acceptor%
                                                                       0456
(savvspc) PROCEDURE(dpa);                                              0503
   %Given the address of a display area, this routine will save the
   viewspecs associated with the area in sysvspec and sysvspec[1]
   (for massaging by setlt).%                                          0504
   %---------------%                                                   0507
   REF dpa;                                                            0508
   sysvspec _ dpa.davspec;                                             0509
   sysvspec[1] _ dpa.davspc2;                                          0510
   RETURN;                                                             0511
   END.
                                                                       0512
(putvspc) PROCEDURE(dpa);                                              0514
   %Given the address of a display area, this routine will put the
   viewspecs saved in sysvspec and sysvspec[1] in the display area.%
                                                                      01375
   %---------------%                                                   0518
   REF dpa;                                                            0519
   (dpa.dapvs, dpa.dapvs2) _ (dpa.davspec, dpa.davspc2);%update
   previous viewspec indicators%                                      01454
   dpa.davspec _ sysvspec;                                             0520
   dpa.davspc2 _ sysvspec[1];                                          0521
   RETURN;                                                             0522
   END.
                                                                       0523
(stklt) PROCEDURE(setchr, vs1, vs2);                                  01656
   %Puts input viewspec characters on stack, appends them to string
   vspstr, and calls setlt to activate them.%                         01657
   %---------------%                                                  01658
   LOCAL vspec[2];                                                    01666
   vspec _ vs1;                                                       01667
   vspec[1] _ vs2;                                                    01668
   IF setchr = BC THEN                                                01617
      BEGIN                                                           01618
      bkc($vspstr);                                                   01619
      POP savevspec TO vspec;                                         01620
```

```
      RETURN(vspec, vspec[1]);                                    01623
      END                                                         01621
   ELSE PUSH vspec ON savevspec;                                  01624
   vspec _ setlt(setchr, vs1, vs2:vspec[1]);                      01665
   *vspstr* _ *vspstr*, setchr;                                   01615
   RETURN(vspec, vspec[1]);                                       01663
   END.
                                                                  01664
   %teedlt and setlt moved to (nls,adrmnp,) 15-OCT-76  %          02245
(softcd)                                                          0801
   %This routine is generally used by the bug accepting routines. It is
   called when they encounter a backspace character while waiting for a
   bug mark.  This routine goes to state if there are no lower entries
   on the stack; otherwise, it pops the return stack, uses DELBM to
   erase the most recent bug mark, and returns.%                  0802
   %---------------%                                             0808
   PROCEDURE;                                                     0809
   delbm();                                                       0812
   RETURN;                                                        0813
   END.
                                                                  01335
(strbug) PROCEDURE(ptr);                                          01946
   %this routine will store the current bug mark in the pointer passed
   it.%                                                           01947
   %-------------%                                               01948
   REF ptr;                                                       01949
   IF bugreg = endfil THEN                                        01950
      ptr _ pbug(lccords() : ptr[1])                              01951
   ELSE                                                           01952
      BEGIN                                                       01953
      ptr _ bugreg;                                               01954
      ptr[1] _ bugreg[1];                                         01955
      END;                                                        01956
   RETURN;                                                        01957
   END.
                                                                  01958
FINISH of PRMSPC L10                                              0861
```

PROC - GUIDE

```
< NLS, PROC-GUIDE.NLS;4, >, 12-OCT-76 09:27 KJM ;;;;
Users' Guide to NLS System Procedures                                  02
   NLS has developed over the years a rich set of procedures for
   manipulating structured files.  These routines constitute a set of
   primitive building blocks with which subsystems can be constructed.
   Subsystems written using them achieve more power with less work than
   do application programs written for linear file systems.  Structured
   files permit parts of a file to be manipulated easily -- e.g. compile
   a branch, mail a group, output a plex.  With this powerful and
   flexible set of procedures, operations on structured files become
   more accessible to the subsystem builder.                           027
   This document is a guide to the user-callable procedures in the NLS
   system.  Most procedures listed here are "class G", which means that
   ARC guarantees they will exist with the same calling conventions and
   semantics.  Subsystem builders who use only these procedures should
   have little or no conversion to do when new versions of NLS are
   brought up.  Conversely, subsystems using lower-level procedures may
   have to be rewritten (possibly substantially) when NLS changes.     024
      The set of procedures listed here may not be complete, in the
      sense that procedures to do some useful operation may not be
      included.  Users are encouraged to suggest that ARC include other
      procedures if they encounter such a situation.                   025
   The procedures described here constitute the "heart" or "core" of
   NLS; hence they are called "core NLS procedures".                   026
   Command branch to generate the procedure guide                     0388
      The following plex of commands can be used to generate the
      procedure descriptions in this file.                            0429
         Execute Programs Delete All <CA>                             0445
         Execute Programs Load Program programs,primitives.rel,<CA>   0449
         Execute Programs Compile Procedure nls,proc-guide,expand<CA>
                                                                      0446
         Execute Programs Institute Program expand<CA> Content <CA>   0447
         Jump Link 0:wi<CA>                                           0448
      (expand)   % U: filter to expand procedure documentation links %
      PROCEDURE (seq % => no value %);                                0457
         % Declarations %                                             0458
            LOCAL stid, doc;                                          0459
            LOCAL TEXT POINTER tp1, tp2, tp3, tp4;                    0460
            LOCAL STRING temp[100];                                   0461
         % See if the statement contains a <link> %                   0462
            IF NOT FIND ^tp1 '< -SP ['>] ^tp2 THEN                    0463
               RETURN (TRUE);                                         0464
         % Expand the link %                                          0465
            stid _ this(seq);                                         0466
            *temp* _ tp1 tp2;                                         0467
            IF NOT doc _ compute(stid, $temp) THEN                    0468
               RETURN (TRUE);                                         0469
         % Replace the link with the procedure description %         0470
            FIND SF(stid) ^tp1 SE(stid) ^tp2 SF(doc) ^tp3 SE(doc) ^tp4;
                                                                      0471
            creptex($tp1, $tp2, $tp3, $tp4);                          0472
            doc _ getsub(doc);                                        0473
            ccopgro(stid, levdown, doc, doc, FALSE, 0);               0474
         RETURN (TRUE);                                               0475
         END.   %%
```

0476
Conventions 03
The following conventions are used in the descriptions of the procedures below. 023
Arguments 016
TEXT POINTER 037
An argument described as type TEXT POINTER is the address of a text pointer. 04
STRING 038
An argument described as type STRING is the address of a string. 05
record type 039
An argument described as a record type is the address of an instance of the record. 06
VIEWSPECS 040
An argument described as type VIEWSPECS is the address of an instance of the viewspecs record. The viewspecs record is described in the NLS Programmers' Guide in the section titled "Additional L10 Capabilities". 035
other types 041
All other argument types (INTEGER, BOOLEAN, STID, ...) are the actual values, not addresses of the values. 07
Values 017
Most procedures return no values or one value. Some group procedures return the head and tail of a group. 022
Signals 018
No core routines trap any signals. However most can generate signals or call routines that generate signals. For example, most core routines generate an error if an argument is illegal, such as passing the stid of a statement which has been deleted. 019
Classification of Procedures 029
Each procedure is classified by its potential for use as a black box (e.g. as a procedure called by user-programs). The following class codes are used to characterize a procedure: 030
G = Guaranteed to stay around with the same user interface for some time to come. 031
U = Useful routine but not guaranteed to stay around from version to version. 032
L = Lower level routine that should only be used when higher level routines will not suffice. Absence of "L" implies a higher level routine. 033
B = NLS backend routine that is callable by any user program. Absence of "B" implies that the routine is not loaded with NLS. 034
Manipulating Text 09
<nls, cedit1, capptex> 0198
<nls, cedit1, ccoptex> 0206
<nls, cedit1, cdeltex> 0214
<nls, cedit1, cinstex> 0218
<nls, cedit1, cmovtex> 0225
<nls, cedit2, creptex> 0231
<nls, cedit2, csetctex> 0237
<nls, cedit2, ctratex> 0245
Manipulating Statements 08
<nls, cedit1, cappsta> 0197
<nls, cedit1, cbresta> 0199