

Ps
--
NE

NE

NE

NE

SR

NNN		NNN	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	AAAAAAAAAA		CCCCCCCCCCCC	PPPPPPPPPP	
NNN		NNN	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	AAAAAAAAAA		CCCCCCCCCCCC	PPPPPPPPPP	
NNN		NNN	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	AAAAAAAAAA		CCCCCCCCCCCC	PPPPPPPPPP	
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	FPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNNNNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNNNNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNNNNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN	NNN	NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCC	PPPPPPPPPP	PPP
NNN	NNN	NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCC	PPPPPPPPPP	PPP
NNN	NNN	NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCC	PPPPPPPPPP	PPP
NNN		NNNNNN	EEE	TTT	AAAAAAAAAAAAAAAA	AAA	CCC	PPP	PPP
NNN		NNNNNN	EEE	TTT	AAAAAAAAAAAAAAAA	AAA	CCC	PPP	PPP
NNN		NNNNNN	EEE	TTT	AAAAAAAAAAAAAAAA	AAA	CCC	PPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCC	PPP	PPP
NNN		NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCCCCCCCCCCC	PPP	PPP
NNN		NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCCCCCCCCCCC	PPP	PPP
NNN		NNN	EEEEEEEEEEEE	TTT	AAA	AAA	CCCCCCCCCCCC	PPP	PPP

```

NN      NN  EEEEEEEEE  TTTTTTTTT  TTTTTTTTT  RRRRRRR  EEEEEEEEE  EEEEEEEEE
NN      NN  EEEEEEEEE  TTTTTTTTT  TTTTTTTTT  RRRRRRR  EEEEEEEEE  EEEEEEEEE
NN      NN  EE          TT          TT          RR      RR  EE          EE
NN      NN  EE          TT          TT          RR      RR  EE          EE
NNNN    NN  EE          TT          TT          RR      RR  EE          EE
NNNN    NN  EE          TT          TT          RR      RR  EE          EE
NN  NN   NN  EEEEEEEEE  TT          TT          RRRRRRR  EEEEEEEEE  EEEEEEEEE
NN  NN   NN  EEEEEEEEE  TT          TT          RRRRRRR  EE,EEEEE  EEEEEEEEE
NN      NN  EE          TT          TT          RR  RR  EE          EE
NN      NN  EE          TT          TT          RR  RR  EE          EE
NN      NN  EE          TT          TT          RR      RR  EE          EE
NN      NN  EE          TT          TT          RR      RR  EE          EE
NN      NN  EEEEEEEEE  TT          TT          RR      RR  EEEEEEEEE  EEEEEEEEE
NN      NN  EEEEEEEEE  TT          TT          RR      RR  EEEEEEEEE  EEEEEEEEE

```

```

LL      IIIIII  SSSSSSS
LL      IIIIII  SSSSSSS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SSSSSS
LL      II     SSSSSS
LL      II     SS
LL      II     SS
LL      II     SS
LL      II     SS
LLLLLLLL  IIIIII  SSSSSSS
LLLLLLLL  IIIIII  SSSSSSS

```

(2)	85	NET\$TRAVERSE_NDI - Traverse NDI collate tree routine
(3)	138	NET\$RESUME_NDI - Resume traversal of NDI collate tree routine
(4)	192	NET\$FIND_NDI - Use collating tree to find an NDI
(4)	193	NET\$FIND_COL - Traverse NDI collate tree to find NDI
(5)	240	NET\$FIND_NAME - Traverse NDI name tree to find NDI
(7)	321	NET\$ADD_NDI - Add entry to collate tree and name tree
(8)	367	ADD_NEW_BTE - Add new entry to AVL tree
(9)	414	INSERT - Recursive routine to insert BTE into TREE
(10)	560	NET\$DELETE_BTE - Delete BTEs from the COLLATE and NAME TREES
(11)	589	DELETE_BTE - Delete element from AVL tree
(13)	745	DEL_REBAL_L - Rebalance after left deletion
(14)	824	DEL_REBAL_R - Rebalance after right deletion
(15)	903	COMPARE_COLLATE - COMPARE BTE DATA ENTRIES BY COLLATING VALUE
(15)	904	COMPARE_COLLATE1 - COMPARE BTE DATA ENTRIES BY COLLATING VALUE

```

0000 1 .TITLE NETTREE - Subroutines for processing BINARY TREES
0000 2 .IDENT 'V04-000'
0000 3 .DEFAULT DISPLACEMENT, LONG
0000 4
0000 5 :*****
0000 6 :*
0000 7 :* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 :* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 :* ALL RIGHTS RESERVED.
0000 10 :*
0000 11 :* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12 :* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13 :* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14 :* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15 :* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16 :* TRANSFERRED.
0000 17 :*
0000 18 :* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19 :* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20 :* CORPORATION.
0000 21 :*
0000 22 :* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23 :* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24 :*
0000 25 :*
0000 26 :*****
0000 27 :
0000 28 :
0000 29 :
0000 30 :++
0000 31 : FACILITY: NETWORK ACP
0000 32 :
0000 33 : ABSTRACT:
0000 34 :
0000 35 :
0000 36 : ENVIRONMENT:
0000 37 :
0000 38 : Kernel mode
0000 39 :
0000 40 : AUTHOR: Rod Gamache, CREATION DATE: 25-Mar-1983
0000 41 :
0000 42 : MODIFIED BY:
0000 43 :
0000 44 : V03-002 RNG0002 Rod Gamache 27-Apr-1983
0000 45 : Skip current NDI entry on scan of NDI database only
0000 46 : if NDI is not the CNR.
0000 47 :
0000 48 : V03-001 RNG0001 Rod Gamache 20-Apr-1983
0000 49 : Stop using the special macros with optional JSBs.
0000 50 :--
0000 51 :
0000 52 :
0000 53 : Library definitions
0000 54 :
0000 55 : $CNFDEF
0000 56 : $CNRDEF
0000 57 : $NFBDEF

```

```
0000 58
0000 59 ;
0000 60 ; Own storage
0000 61 ;
00000000 62 .PSECT NET_IMPURE,WRT,NOEXE
0000 63
00000004 0000 64 NAME_BUF:.BLKL 1 ; String storage for executor name
0004 65
00000000 0000 66 .PSECT NET_CODE1,NOWRT,EXE
0004 67
0000 68 ; Local definitions
0000 69 ;
0000 70 $DEFINI BTE GLOBAL ; Define Binary Tree Elements
0000 71
0000 72 $DEF BTESL_LEFT .BLKL 1 ; Left subtree pointer
0004 73 $DEF BTESL_RIGHT .BLKL 1 ; Right subtree pointer
0008 74 $DEF BTESW_SIZE .BLKW 1 ; Size of BTE structue
000A 75 $DEF BTESB_TYPE .BLKB 1 ; Structue type (FF or FE)
000B 76 $DEF BTESB_BAL .BLKB 1 ; Subtree balance (-1, 0 or +1)
000C 77 $DEF BTESL_BTEPTR .BLKL 1 ; Pointer to node in other AVL tree
0010 78 $DEF BTESL_PTR .BLKL 1 ; Pointer to NDI database
0014 79 $DEF BTEST_DATA .BLKB 15+3+1 ; BTE data CIRCUIT NAME + ADDRESS +
0027 80 ; COUNT
00000013 0027 81 BTESC_DATA_SIZE = <.-BTEST_DATA> ; Size of data portion of BTE
0027 82 $DEF BTESC_LENGTH ; Size of BTE
0027 83 $DEFEND BTE
```

```

0000 85      .SBTTL NET$TRAVERSE_NDI - Traverse NDI collate tree routine
0000 86      :++
0000 87      : NET$TRAVERSE_NDI - iraverse NDI collate tree routine
0000 88      :
0000 89      : This routine traverses the collate tree, calling back the caller with
0000 90      : the next node address. The search is done in inorder fashion.
0000 91      :
0000 92      : INPUTS:
0000 93      :
0000 94      :     R11 = Address of CNR
0000 95      :     R10 = Address of start CNF (or zero if start at begining)
0000 96      :
0000 97      : OUTPUTS:
0000 98      :
0000 99      :     R10 = Address of CNF if found, else same as on input.
0000 100     :     R0 = True if success, else false
0000 101     :
0000 102     :     R1,R2 are destroyed
0000 103     :
0000 104     :--
0000 105     .ENABL  LSB
0000 106     NET$TRAVERSE_NDI::
0000 107     MOVL   (SP),R1           ; Co-routine to traverse the tree
0000 108     MOVL   NET$GL_COL_TREE,R2 ; Save return address
0000 109     TSTL   R10              ; Assume we should start at begining
0000 110     BEQL   TRAVERSE        ; Should we start at begining?
0000 111     ; Br if yes
0000 112     ASSUME CNF$$_COLBTE EQ CNR$$_COLBTE
0000 113     MOVL   CNF$$_COLBTE(R10),R2 ; Copy start BTE address
0000 114     BEQL   90$            ; Br if none
0000 115     ;
0000 116     TRAVERSE:           ; Recursive to traverse the tree
0000 117     TSTL   R2              ; Bottom of tree?
0000 118     BEQL   90$            ; Br if yes
0000 119     PUSHL R2              ; Save last BTE address
0000 120     MOVL   BTE$$_LEFT(R2),R2 ; Get address of left subtree
0000 121     BSBB   TRAVERSE        ; Traverse the tree
0000 122     RESUME_CONT:
0000 123     POPL   R2              ; Restore previous BTE address
0000 124     MOVL   BTE$$_PTR(R2),R10 ; Get the CNF address of the NDI
0000 125     MOVZBL #1,R0          ; Indicate more to come
0000 126     JSB   (R1)            ; Call back caller with CNF in R10
0000 127     ;
0000 128     NET$TRAVERSE_ALT::  ; Alternate entry point, stack reset
0000 129     MOVL   (SP)+,R1       ; Pop return address from stack
0000 130     MOVL   BTE$$_RIGHT(R2),R2 ; Point to right subtree
0000 131     BNEQ   TRAVERSE        ; Traverse left subtree of right half
0000 132     90$: CLRL   R0        ; Indicate no more
0000 133     RSB
0000 134     ;
0000 135     .DSABL  LSB
0000 136

```

```

0036 138 .SBTTL NET$RESUME_NDI - Resume traversal of NDI collate tree routine
0036 139 :++
0036 140 : NET$RESUME_NDI - Resume traversal of NDI collate tree routine
0036 141 :
0036 142 : This routine traverses the collate tree, looking for the CNF address that
0036 143 : was given. On it's way, it builds a new stack just as if it was in the
0036 144 : middle of the co-routine NET$TRAVERSE NDI. This way, we can resume where
0036 145 : we left off. If the tree element is not found then the stack is left in
0036 146 : tack so we can proceed, this is in case we are deleting the BTEs.
0036 147 :
0036 148 : INPUTS:
0036 149 :
0036 150 : R11 = Address of CNR
0036 151 : R10 = Address of start CNF, cannot be zero
0036 152 : R7,R8 = Descriptor of collating string
0036 153 :
0036 154 : OUTPUTS:
0036 155 :
0036 156 : R0 = Bit 0: True if element found or we can proceed, else false
0036 157 : Bit 1: Clear if we must proceed to next, else take current
0036 158 :
0036 159 : R1,R2 are destroyed
0036 160 :
0036 161 :--
0036 162 .ENABL LSB
52 00000000'EF 50 D0 0036 163 NET$RESUME_NDI:: : Routine to traverse the tree
: Save return address
: Start at begining
: Assume failure
0036 164 MOVL (SP)+,R1
0036 165 MOVL NET$GL_COL_TREE,R2
0036 166 CLRL R0
0042 167 RESUME: : Recursive routine to traverse tree
: Bottom of tree?
: Br if yes, error
: Compare key on current node
: Br if to the right
: Br if found, leave
: Save current BTE address
: Get address of left subtree
: Push address of continuation
: Indicate success, something on stack
: Traverse the tree
: Point to right subtree
: Traverse left subtree of right half
: Br if nothing on stack, return error
: Pop last address from stack
: Else, pick up last BTE on stack
: Take the current BTE
: Set flag to take the current BTE
: Indicate success
: Return to caller
0042 168 TSTL R2
0044 170 BEQL 70$
0046 171 BSBW COMPARE_COLLATE1
0049 172 BGTRU 30$
0048 173 BEQL 90$
004D 174 PUSHL R2
004F 175 MOVL BTE$L_LEFT(R2),R2
0052 176 PUSHAB RESUME_CONT
0055 177 MOVZBL #1,R0
0058 178 BRB RESUME
005A 179 30$: MOVL BTE$L_RIGHT(R2),R2
005E 180 BRB RESUME
0060 181 70$: BLBC R0,100$
0063 182 ADDL #4,SP
0066 183 MOVL (SP)+,R2
0069 184 MOVL BTE$L_PTR(R2),R10
006D 185 SETBIT #1,R0
0071 186 90$: SETBIT #0,R0
0075 187 100$: JMP (R1)
0077 188
0077 189 .DSABL LSB
0077 190

```

```

0077 192      .SBTTL NET$FIND_NDI - Use collating tree to find an NDI
0077 193      .SBTTL NET$FIND_COL - Traverse NDI collate tree to find NDI
0077 194      ;++
0077 195      ; NET$FIND_NDI - Use collating tree to find an NDI
0077 196      ; NET$FIND_COL - Traverse NDI collate tree to find an NDI
0077 197      ;
0077 198      ; This routine traverses the collate tree, searching for a match on the input
0077 199      ; key value.
0077 200      ;
0077 201      ; INPUTS:
0077 202      ;
0077 203      ; R11 = Address of CNR
0077 204      ; R10 = Address of CNF to start searching from
0077 205      ; R7,R8 = Descriptor for key value
0077 206      ;
0077 207      ; OUTPUTS:
0077 208      ;
0077 209      ; R10 = Address of CNF found else the starting CNF.
0077 210      ; R3 = Address of BTE who's key was LSSU but closest to the given key.
0077 211      ; R4 = Address of BTE who's key was GTRU but closest to the given key,
0077 212      ; or zero if end of tree.
0077 213      ; R0 = True if success, else false
0077 214      ;
0077 215      ;--
0077 216      ;
0077 217      NET$FIND_NDI:: ; Use the collating tree to find an NDI
0077 218      NET$FIND_COL:: ; Traverse the Collate tree for a match
0077 219      NET$FIND_NDI:: ; Save registers
0077 220      PUSHL R9 ; Assume failure
0077 221      CLRL R0 ; Assume we start from begining
0077 222      MOVL NET$GL_COL_TREE,R9 ; Br if no root, yet
0077 223      BEQL 90$ ; Start from begining?
0077 224      TSTL R10 ; Br if yes
0077 225      BEQL 50$ ; Assume failure
0077 226      CLRL R0 ;
0077 227      ASSUME CNF$$_COLBTE EQ CNR$$_COLBTE
0077 228      MOVL CNF$$_COLBTE(R10),R9 ; Else, pick up starting BTE
0077 229      BEQL 90$ ; Br if no BTE
0077 230      CMPL R10,R11 ; Is this the CNR?
0077 231      BEQL 50$ ; Br if yes, don't skip first BTE
0077 232      MOVL BTE$$_RIGHT(R9),R9 ; Else, skip current and continue
0077 233      CLRL R4 ; Assume end of tree
0077 234      BSBB FIND ; Find a match
0077 235      BLBC R0,90$ ; Br if failure
0077 236      MOVL BTE$$_PTR(R9),R10 ; Else, get the CNF address
0077 237      POPL R9 ; Restore registers
0077 238      RSB ; Return to caller

```



```

00A7 240      .SBTTL NET$FIND_NAME - Traverse NDI name tree to find NDI
00A7 241      :++
00A7 242      : NET$FIND_NAME - Traverse NDI name tree to find an NDI
00A7 243      :
00A7 244      : This routine traverses the name tree, searching for a match on the input
00A7 245      : key value.
00A7 246      :
00A7 247      : INPUTS:
00A7 248      :
00A7 249      :     R11 = Address of CNR
00A7 250      :     R10 = Address of CNF to start searching from
00A7 251      :     R7,R8 = Descriptor for key value
00A7 252      :
00A7 253      : OUTPUTS:
00A7 254      :
00A7 255      :     R10 = Address of CNF found else the starting CNF address
00A7 256      :     R0 = True if success, else false
00A7 257      :
00A7 258      :--
00A7 259
00A7 260 NET$FIND_NAME::      : Traverse the name tree for a match
59 0218 8F BB 00A7 261      PUSHR   #^M<R3,R4,R9>      : Save registers
00000000 EF D0 00AB 262      MOVL   NET$GL_NAME_TREE,R9      : Assume we start from begining
5A D5 00B2 263      TSTL   R10      : Start from begining?
11 13 00B4 264      BEQL   50$      : Br if yes
50 D4 00B6 265      CLRL   R0      : Assume failure
00B8 266
00B8 267      ASSUME  CNF$&L_COLBTE EQ CNR$&L_COLBTE
59 04 AA D0 00B8 268      MOVL   CNF$&L_NAMEBTE(R10),R9      : Else, pick up starting BTE
12 13 00BC 269      BEQL   90$      : Br if no BTE
5B 5A D1 00BE 270      CMPL   R10,R11      : Is this the CNR?
04 13 00C1 271      BEQL   50$      : Br if yes, don't skip first BTE
59 04 A9 D0 00C3 272      MOVL   BTE$&L_RIGHT(R9),R9      : Else, skip current and continue
0C 10 00C7 273 50$:      BSBB   FIND      : Find a match
04 50 E9 00C9 274      BLBC   R0,90$      : Br if failure
5A 10 A9 D0 00CC 275      MOVL   BTE$&L_PTR(R9),R10      : Else, get the CNF address
0218 8F BA 00D0 276 90$:      POPR   #^M<R3,R4,R9>      : Restore registers
00D4 277      RSB      : Return to caller

```

```

00D5 279
00D5 280 :+
00D5 281 : Recursive routine to find an entry in the collated tree. This routine
00D5 282 : will return the address of the BTE that is closest to the request key value,
00D5 283 : but less than the requested key value.
00D5 284 :
00D5 285 : Inputs:
00D5 286 : R9 = Address of first BTE
00D5 287 : R7,R8 = Descriptor for key value
00D5 288 :
00D5 289 : Outputs:
00D5 290 : R3 = Address of BTE who's key was LSSU but closest to the given key.
00D5 291 : R4 = Address of BTE who's key was GTRU but closest to the given key,
00D5 292 : or zero if end of tree.
00D5 293 :
00D5 294 : -
00D5 295 FIND: ; Recursive routine to find a BTE
59 D5 00D5 296 TSTL R9 ; End of tree?
03 12 00D7 297 BNEQ 10$ ; Br if no
50 D4 00D9 298 CLRL R0 ; Else, return failure
05 00DB 299 RSB ; Return to caller
00DC 300
037B 30 00DC 301 10$: BSBW COMPARE_COLLATE ; Compare key on current node
0C 1A 00DF 302 BGTRU 50$ ; Br if to the right
50 04 1F 00E1 303 BLSSU 30$ ; Br if to the left
01 9A 00E3 304 MOVZBL #1,R0 ; Indicate success
05 00E6 305 RSB
00E7 306
00E7 307 ; Search the left subtree
00E7 308
59 69 D0 00E7 309 30$: MOVL BTE$L_LEFT(R9),R9 ; Point to left subtree
E9 10 00EA 310 BSBB ; Keep trying
05 00EC 311 RSB ; All done
00ED 312
00ED 313 ; Search the right subtree
00ED 314
53 59 D0 00ED 315 50$: MOVL R9,R3 ; Save LSSU BTE address
59 04 A9 D0 00F0 316 MOVL BTE$L_RIGHT(R9),R9 ; Point to right subtree
54 59 D0 00F4 317 MOVL R9,R4 ; Save address of current for GTRU BTE
DC 10 00F7 318 BSBB ; Keep looking
05 00F9 319 RSB
  
```

```

00FA 321      .SBTTL NET$ADD_NDI - Add entry to collate tree and name tree
00FA 322      :++
00FA 323      : NET$ADD_NDI - Add an entry to the COLLATE TREE and the NAME TREE
00FA 324      :
00FA 325      : This routine adds a node address to the collate tree, and at the
00FA 326      : same time it adds a name entry to the name tree.
00FA 327      :
00FA 328      : INPUTS:
00FA 329      :
00FA 330      :     R11 = Address of CNR
00FA 331      :     R10 = Address of CNF to be added
00FA 332      :
00FA 333      : OUTPUTS:
00FA 334      :
00FA 335      :     R0 = True if success, else false
00FA 336      :
00FA 337      :--
00FA 338
00FA 339 NET$ADD_NDI::
03FE 8F BB 00FA 340      PUSHR   #^M<R1,R2,R3,R4,R5,R6,R7,R8,R9> ; Save registers
00FE 341      $GETFLD ndi,s,col ; Get the collating value
57 12 91 010B 342      CMPB    #BTESC_DATA_SIZE-1,R7 ; Check the string size
04 18 010E 343      BGEQ    20$ ; Br if okay
0110 344      BUG_CHECK NETNOSTATE,FATAL ; Else, bug check
0114 345
59 00000000'EF D0 0114 346 20$: MOVL   NET$GL_COL_TREE,R9 ; Get address of COLLATE tree root
59 10 01 B 347      BSBB    ADD_NEW_BTE ; Add new BTE to COLLATE tree
00000000'EF 59 D0 011D 348      MOVL   R9,NET$GL_COL_TREE ; Set new root of tree
56 DD 0124 349      PUSHL  R6 ; Save address of BTE
0126 350      $GETFLD ndi,s,na ; Get the node name descriptor
11 50 E8 0133 351      BLBS   R0,40$ ; Br if present
58 00000000'EF 9E 0136 352      MOVAB  NAME_BUF,R8 ; Get address of temporary name buffer
68 2D2D2D2D 8F D0 013D 353      MOVL   #'A'-----',(R8) ; Use fictitious name (for exec)
57 04 9A 0144 354      MOVZBL #4,R7 ; Set size of string
59 00000000'EF D0 0147 355 40$: MOVL   NET$GL_NAME_TREE,R9 ; Get address of NAME tree root
26 10 014E 356      BSBB    ADD_NEW_BTE ; Add new entry to NAME tree
00000000'EF 59 D0 0150 357      MOVL   R9,NET$GL_NAME_TREE ; Set new root of tree
0A A6 01 90 0157 358      MOVB   #1,BTESB_TYPE(R6) ; Set different structure type
55 8ED0 015B 359      POPL   R5 ; Restore address of COLLATE BTE
0C A5 56 D0 015E 360      MOVL   R6,BTESL_BTEPTR(R5) ; Set the cross pointers
0C A6 55 D0 0162 361      MOVL   R5,BTESL_BTEPTR(R6) ;
6A 55 D0 0166 362      MOVL   R5,CNF$COLBTE(R10) ; Set pointers in CNF to tree structs.
04 AA 56 D0 0169 363      MOVL   R6,CNF$NAMEBTE(R10) ;
03FE 8F BA 016D 364 90$: POPR   #^M<R1,R2,R3,R4,R5,R6,R7,R8,R9> ; Restore registers
05 0171 365      RSB ; Exit with status

```

```

0172 367      .SBTTL  ADD_NEW_BTE - Add new entry to AVL tree
0172 368 :++
0172 369 : ADD_NEW_BTE - Add new entry to AVL tree
0172 370 :
0172 371 : INPUTS:
0172 372 :
0172 373 :     R10 = Address of CNF to be added
0172 374 :     R9 = Root of tree to insert onto or zero if no root yet
0172 375 :     R7,R8 = Descriptor for key value
0172 376 :
0172 377 : OUTPUTS:
0172 378 :
0172 379 :     Entry inserted into correct position of collate tree.
0172 380 :
0172 381 :     R9 = Address of new root of tree
0172 382 :     R6 = Address of BTE added.
0172 383 :     R1-R5 are destroyed.
0172 384 :--
0172 385 ASSUME  BTE$$_LEFT  EQ  0
0172 386 ASSUME  BTE$$_RIGHT EQ  4+BTE$$_LEFT
0172 387 ASSUME  BTE$$_SIZE EQ  4+BTE$$_RIGHT
0172 388 ASSUME  BTE$$_TYPE EQ  2+BTE$$_SIZE
0172 389 ASSUME  BTE$$_BAL  EQ  1+BTE$$_TYPE
0172 390 ASSUME  BTE$$_BTEPTR EQ 1+BTE$$_BAL
0172 391 ASSUME  BTE$$_PTR   EQ  4+BTE$$_BTEPTR
0172 392 ASSUME  BTE$$_DATA EQ  4+BTE$$_PTR
0172 393
0172 394 BUG_OUT:
0172 395 BUG_CHECK NETNOBUF,FATAL      ; Fatal error
0172 396
0172 397 ADD_NEW_BTE:
0172 398      MOVZBL #BTE$$_LENGTH,R1      ; Add new entry to AVL tree
51  27  9A 0176 398      JSB NET$$_ALLOCATE           ; Set size of needed BTE
00000000'EF 16 0179 399      BLBC R0,BUG_OUT           ; Allocate a BTE
FO 50  E9 017F 400      MOVL R2,R6           ; Bug check on failure
56  52  D0 0182 401      CLRQ (R2)+           ; Point R6 to new BTE
82  82  7C 0185 402      MOVL R1,(R2)+           ; Clear pointers
82  51  D0 0187 403      CLRL (R2)+           ; Set size of BTE, TYPE 0, clear BAL
82  82  D4 018A 404      MOVL R10,(R2)+          ; Clear BTEPTR
82  SA  D0 018C 405      MOVL R7,(R2)+          ; Set PTR
82  57  90 018F 406      MOVV R7,(R2)+          ; Set size of string
62  68  57  28 0192 407      MOVC3 R7,(R8),(R2) ; Store the collating value
0196 408
0196 409 :     INSERT ENTRY INTO ASCENDING VALUE ORDERED TREE
0196 410 :
0196 411 :: BRB  INSERT                  ; Insert new entry
0196 412
  
```

NET
 Sym
 ADD
 BTE
 BTE
 BTE
 BTE
 BTE
 BTE
 BTE
 BTE
 BTE
 BUG
 BUG
 BUG
 CNF
 CNF
 CNF
 CNF
 CNR
 COM
 COM
 DEL
 DEL
 DEL
 DEL
 DLR
 DLR
 DLR
 DLR
 DLR
 DRL
 DRL
 DRL
 FIN
 INS
 NAM
 NET
 NET
 NET
 NET
 NET
 NET
 NET
 NET
 NET
 NET
 NET
 NFI
 NET
 NET
 NFB
 NFB
 RES
 RES
 TRA
 _\$\$

```

0196 414 .SBTTL INSERT - Recursive routine to insert BTE into TREE
0196 415 :++
0196 416 : INSERT - Recursive routine to insert BTE into TREE
0196 417 :
0196 418 : This routine is called to insert the entry for a given key into
0196 419 : an AVL tree. It returns the appropriate tree node, then
0196 420 : rebalances the tree as required. For a full explanation of
0196 421 : what is involved, see WIRTH, 'Algorithm + Data Structures = Programs'.
0196 422 : Basically, we scan the tree until we find the node with the
0196 423 : given key. We then look for the rightmost decendent of the
0196 424 : the left subtree and delete that entry, putting its associated value
0196 425 : (pointer) in the original node. This avoids a lot of node shuffling.
0196 426 : Then, we go up the tree until we find a node which remains in
0196 427 : acceptable balance when the subtree gets lower.
0196 428 :
0196 429 : INPUTS:
0196 430 :
0196 431 : R9 = Pointer to current node in TREE
0196 432 : R7,R8 = Descriptor of Key value for insertion
0196 433 : R6 = Pointer to new BTE entry
0196 434 :
0196 435 : OUTPUTS:
0196 436 :
0196 437 : R9 = Address of new root of tree
0196 438 : R6 = New pointer to current node in tree
0196 439 : R0-R2 are destroyed.
0196 440 :--
0196 441 :
0196 442 INSERT:
59 07 D5 0196 443 TSTL R9 ; Recursive routine to insert
0198 444 BNEQ 10$ ; Null pointer? (BOTTOM OF TREE)
019A 445 ; ; Br if not
019A 446 ; REACHED BOTTOM OF TREE - INSERT INTO TREE AT BOTTOM
019A 447 ;
59 50 56 01 50 D0 019A 448 MOVL R6,R9 ; Set address of new node
019D 449 MOVL #1,R0 ; Indicate not balanced
01A0 450 RSB
01A1 451 ;
01A1 452 ; CHECK IF KEY IS TO THE LEFT OR RIGHT SUBTREE
01A1 453 ;
02B6 07 30 01A1 454 10$: BSBW COMPARE_COLLATE ; Compare key on current node
01A4 455 BGTRU 200$ ; Br if to the right side
01A6 456 :: BEQL 300$ ; Br if already exists
01A6 457 ;
01A6 458 ; INSERT THE NODE INTO THE LEFT SUBTREE
01A6 459 ;
59 69 DD 01A6 460 PUSHL R9 ; Save pointer of father
59 69 DO 01A8 461 MOVL BTE$LEFT(R9),R9 ; Setup pointer to left son
51 59 E9 10 01A8 462 BSBB INSERT ; Insert into left subtree
51 59 DO 01AD 463 MOVL R9,R1 ; Save new address of subtree
69 59 BEDO 01B0 464 POPL R9 ; Return to father node
69 51 DO 01B3 465 MOVL R1,BTE$LEFT(R9) ; Point to new left subtree
06 50 E8 01B6 466 BLBS R0,110$ ; Br if not balanced
00B9 31 01B9 467 40$: BRW 90$ ; Exit
00B4 31 01BC 468 50$: BRW 80$ ; Exit - mark in balance
01BF 469 ;
01BF 470 ; THE LEFT SUBTREE HAS GROWN HIGHER - RESTORE BALANCE

```

NE
PS

PSE

\$AE
NE
NE

Ph

In
Co
Pas
Syn
Pas
Syn
Pse
Cro
Ass

The
44E
The
942
18

Mac

-S
-S
-S
-S
-S
-S
TO
69
The
MA

```

01BF 471
OB A9 97 01BF 472 110$: DECB BTE$B_BAL(R9) ; Perform left shift of tree
F8 13 01C2 473 BEQL 50$ ; Exit if in perfect balance
F1 OB A9 E8 01C4 474 BLBS BTE$B_BAL(R9),40$ ; Exit if avl balanced
51 69 D0 01C8 475 MOVL BTE$L_LEFT(R9),R1 ; Get pointer to left subtree
OB A1 95 01CB 476 TSTB BTE$B_BAL(R1) ; Test balance on that side
11 18 01CE 477 BGEQ 120$ ; Br if need double rotation
01D0 478
01D0 479 ; PERFORM SINGLE LL ROTATION
01D0 480
69 04 A1 D0 01D0 481 MOVL BTE$L_RIGHT(R1),BTE$L_LEFT(R9)
04 A1 59 D0 01D4 482 MOVL R9,BTE$L_RIGHT(R1)
OB A9 94 01D8 483 CLRB BTE$B_BAL(R9) ; Mark in balance
59 51 D0 01DB 484 MOVL R1,R9 ; Set new root of subtree
008F 31 01DE 485 BRW 250$ ; Indicate in perfect balance
01E1 486
01E1 487 ; PERFORM DOUBLE LR ROTATION
01E1 488
52 04 A1 D0 01E1 489 120$: MOVL BTE$L_RIGHT(R1),R2 ; Get right son of left subtree
04 A1 62 D0 01E5 490 MOVL BTE$L_LEFT(R2),BTE$L_RIGHT(R1)
62 51 D0 01E9 491 MOVL R1,BTE$L_LEFT(R2)
69 04 A2 D0 01EC 492 MOVL BTE$L_RIGHT(R2),BTE$L_LEFT(R9)
04 A2 59 D0 01F0 493 MOVL R9,BTE$L_RIGHT(R2)
OB A9 94 01F4 494 CLRB BTE$B_BAL(R9) ; Initialize balance indicators
OB A1 94 01F7 495 CLRB BTE$B_BAL(R1)
OB A2 95 01FA 496 TSTB BTE$B_BAL(R2) ; Check balance of new subtree
6E 13 01FD 497 BEQL 240$ ; Skip if in balance
06 14 01FF 498 BGTR 130$ ; Br if left side heavy
OB A9 01 90 0201 499 MOVB #1,BTE$B_BAL(R9) ; Mark right side heavy
66 11 0205 500 BRB 240$ ; Set new subtree and exit
OB A1 01 8E 0207 501 130$: MNEGB #1,BTE$B_BAL(R1) ; Mark left side heavy
60 11 020B 502 BRB 240$ ; Set new subtree and exit
020D 503
020D 504 ; INSERT NEW NODE INTO RIGHT SUBTREE
020D 505
59 59 DD 020D 506 200$: PUSHL R9 ; Save pointer of father
04 A9 D0 020F 507 MOVL BTE$L_RIGHT(R9),R9 ; Setup pointer to right son
FF80 30 0213 508 BSBW INSERT ; Insert into left subtree
51 59 D0 0216 509 MOVL R9,R1 ; Save new address of subtree
59 BED0 0219 510 POPL R9 ; Return to father node
04 A9 51 D0 021C 511 MOVL R1,BTE$L_RIGHT(R9) ; Point to new right subtree
52 50 E9 0220 512 BLBC R0,90$ ; Exit if balanced
0223 513
0223 514
0223 515 ; THE RIGHT SUBTREE HAS GROWN HIGHER - RESTORE BALANCE
OB A9 96 0223 516 INCB BTE$B_BAL(R9) ; Perform right shift of tree
49 OB 4B 13 0226 517 BEQL 80$ ; Exit if in perfect balance
OB A9 E8 0228 518 BLBS BTE$B_BAL(R9),90$ ; Exit if avl balanced
51 04 A9 D0 022C 519 MOVL BTE$L_RIGHT(R9),R1 ; Get pointer to right subtree
OB A1 95 0230 520 TSTB BTE$B_BAL(R1) ; Test balance on that side
OF 15 0233 521 BLEQ 220$ ; Br if need double rotation
0235 522
0235 523 ; PERFORM SINGLE RR ROTATION
0235 524
04 A9 61 D0 0235 525 MOVL BTE$L_LEFT(R1),BTE$L_RIGHT(R9)
61 59 DC 0239 526 MOVL R9,BTE$L_LEFT(R1)
OB A9 94 023C 527 CLRB BTE$B_BAL(R9) ; Mark in balance

```

```

59 51 D0 023F 528      MOVL  R1,R9          ; Set new root of subtree
      2C 11 0242 529      BRB   250$          ; Indicate in perfect balance
      0244 530          :
      0244 531          : PERFORM DOUBLE RL ROTATION
      0244 532          :
61 52 61 D0 0244 533 220$: MOVL  BTESL_LEFT(R1),R2      ; Get left son of right subtree
04 A2 04 A2 D0 0247 534      MOVL  BTESL_RIGHT(R2),BTESL_LEFT(R1)
04 A9 62 D0 024B 535      MOVL  R1,BTESL_RIGHT(R2)
      62 59 D0 024F 536      MOVL  BTESL_LEFT(R2),BTESL_RIGHT(R9)
      OB A9 94 0253 537      MOVL  R9,BTESL_LEFT(R2)
      OB A9 94 0256 538      CLRB  BTESB_BAL(R9)          ; Initialize balance indicators
      OB A1 94 0259 539      CLRB  BTESB_BAL(R1)
      OB A2 95 025C 540      TSTB  BTESB_BAL(R2)          ; Check balance of new subtree
      OC 13 025F 541      BEQL  240$          ; Skip if in balance
      06 14 0261 542      BGTR  230$          ; Br if left side heavy
OB A1 01 90 0263 543      MOVB  #1,BTESB_BAL(R1)      ; Mark right side heavy
      04 11 0267 544      BRB   240$          ; Set new subtree and exit
OB A9 01 8E 0269 545 230$: MNEGB #1,BTESB_BAL(R9)      ; Mark left side heavy
      026D 546          :
      026D 547          : SET NEW POINTER TO SUBTREE
      026D 548          :
59 52 D0 026D 549 240$: MOVL  R2,R9          ; Set new subtree
      0270 550          :
      0270 551          : MARK SUBTREE IN PERFECT BALANCE
      0270 552          :
      OB A9 94 0270 553 250$: CLRB  BTESB_BAL(R9)          ; Mark in perfect balance
      0273 554          :
      0273 555          : RETURN R0 FALSE TO INDICATE NEITHER SIDE IS HEAVY
      0273 556          :
      50 D4 0273 557 80$: CLRL  R0          ; Mark in balance
      05 0275 558 90$: RSB
  
```

```

0276 560 .SBTTL NET$DELETE_BTE - Delete BTEs from the COLLATE and NAME TREES
0276 561 :++
0276 562 : NET$DELETE_BTE - Delete BTEs from the COLLATE and NAME TREES
0276 563 :
0276 564 : This routine deletes the collate and name BTEs from the corresponding AVL
0276 565 : trees.
0276 566 :
0276 567 : INPUTS:
0276 568 :
0276 569 : R11 = Address of CNR
0276 570 : R10 = Address of CNF to be deleted
0276 571 :
0276 572 : OUTPUTS:
0276 573 :
0276 574 : R0 = True if success, else false
0276 575 :
0276 576 : R9 is destroyed.
0276 577 :
0276 578 :--
0276 579
0276 580 NET$DELETE_BTE:: : Delete the BTEs for collate and names
0276 581 PUSHQ R7 : Save registers
0279 582 $GETFLD ndi,s,col : Get the collating value
0286 583 MOVQ R7,-(SP) : Push descriptor of key value
0289 584 PUSHAL NET$GL_COL_TREE : Push address of collate tree
028F 585 CALLS #3,DELETE_BTE : Delete the BTEs
0296 586 POPQ R7 : Restore registers
05 0299 587 RSB : Exit with status

```

```

7E 57 7D
00000000'EF DF
0000029A'EF 03 FB

```



```

029A 589      .SBTTL DELETE_BTE - Delete element from AVL tree
029A 590      :++
029A 591      : DELETE_BTE - Delete an element from an AVL tree
029A 592      :
029A 593      : This routine is called to delete the entry for a given key from
029A 594      : an AVL tree. It returns the appropriate tree node, then
029A 595      : rebalances the tree as required. For a full explanation of
029A 596      : what is involved, see WIRTH, 'Algorithm + Data Structures = Programs'.
029A 597      : Basically, we scan the tree until we find the node with the
029A 598      : given key. We then look for the rightmost decendent of the
029A 599      : the left subtree and delete that entry, putting its associated value
029A 600      : (pointer) in the original node. This avoids a lot of node shuffling.
029A 601      : Then, we go up the tree until we find a node which remains in
029A 602      : acceptable balance when the subtree gets lower.
029A 603      :
029A 604      : INPUTS:      4(AP)  Address of root of tree
029A 605      :              8(AP)  Length of string for key value
029A 606      :              12(AP) Address of string for key value
029A 607      :
029A 608      : OUTPUTS:
029A 609      :              R0      True if node in tree, else false
029A 610      :
029A 611      : SIDE EFFECTS:
029A 612      :
029A 613      : Root address of tree possibly updated.
029A 614      : Node deleted from tree, if present.
029A 615      :--
029A 616      :
029A 617      DELETE_BTE:      ; Delete a binary tree element (BTE)
029A 618      .WORD      ^M<R1,R2,R3,R4,R5,R6,R7,R8,R9> ; Save registers
029A 619      CLRL      R0      ; Assume failure
59 04 BC 50 03FE 029C 620      MOVL      @4(AP),R9      ; Get address of first BTE
029E 621      BEQL      10$      ; Br if none, exit with error
57 08 AC 0F 13 02A2 622      MOVQ      8(AP),R7      ; Get value to pass to comparison routine
02A4 623      CLRL      R6      ; Indicate node not found yet
02A8 624      BSBB      DELETE      ; Call routine which does the work
04 BC 50 08 1C 02AA 625      MOVL      R1,@4(AP)      ; Set new root of tree
50 01 51 04 02AC 626      MOVZBL  #1,R0      ; Return success
02B0 627      RET      ; Exit with status
02B3 628      10$:
02B4

```

```

02B4 630 :+
02B4 631 :+ Auxiliary routine to go down the tree recursively. If it finds
02B4 632 :+ the matching node, it passes its address down. On the way up,
02B4 633 :+ this routine rebalances the tree.
02B4 634 :+
02B4 635 : Inputs: R9 Address of node in tree to start searching from
02B4 636 : R7,R8 Descriptor of parameter to pass to comparison routine
02B4 637 : R6 Address of node that matched or zero
02B4 638 :
02B4 639 : Outputs:
02B4 640 : R2 Change in height indicator (zero if no change, else -1)
02B4 641 : R1 Address of new parent for tree/subtree
02B4 642 :-
02B4 643 :-
02B4 644 DELETE:
02B4 645 :
02B4 646 : Call the comparison routine to check for match
02B4 647 : R9 Address of node to check
02B4 648 : R7,R8 Descriptor of key value
02B4 649 :
01A3 30 02B4 650 BSBW COMPARE_COLLATE ; Call compare routine
2C 1A 02B7 651 BGTRU 40$ ; Br if key was greater, go to right
03 1F 02B9 652 BLSSU 20$ ; Br if key was less, go to left
02BB 653 :
02BB 654 : Found match, save node address
56 59 D0 02BB 655 :
02BE 656 20$: MOVL R9,R6 ; Else remember that this is the node
02BE 657 :
02BE 658 : See if we can traverse the left subtree
02BE 659 :
50 69 D0 02BE 660 MOVL BTESL_LEFT(R9),R0 ; Get left subtree
06 12 02C1 661 BNEQ 30$ ; Br if there is one
51 04 A9 D0 02C3 662 MOVL BTESL_RIGHT(R9),R1 ; Else get right subtree
25 11 02C7 663 BRB 50$ ; And go to common code
02C9 664 :
02C9 665 : Traverse left subtree, with subtree present
02C9 666 :
59 59 DD 02C9 667 3J$: PUSHL R9 ; Save current BTE
59 50 D0 02CB 668 MOVL R0,R9 ; Get subtree pointer
E4 10 02CE 669 BSBB DELETE ; Make recursive call (finds leaf)
59 8ED0 02D0 670 POPL R9 ; Get current BTE back
69 51 D0 02D3 671 MOVL R1,BTESL_LEFT(R9) ; And set new subtree ptr
0B A9 52 82 02D6 672 SUBB R2,BTESB_BAL(R9) ; Adjust balance
52 95 02DA 673 TSTB R2 ; Any change in height?
04 12 02DC 674 BNEQ 35$ ; Br if yes - rebalance
51 59 D0 02DE 675 MOVL R9,R1 ; Set new parent (leave R2 = 0)
05 02E1 676 RSB
0081 31 02E2 677 :
02E2 678 35$: BRW DEL_REBAL_L ; Else, go and rebalance
02E5 679 :
02E5 680 40$: :
02E5 681 :
02E5 682 : See if we can traverse right subtree
50 04 A9 D0 02E5 683 MOVL BTESL_RIGHT(R9),R0 ; Get right subtree
5D 12 02E9 684 BNEQ 60$ ; Br if there is one
51 69 D0 02EB 685 MOVL BTESL_LEFT(R9),R1 ; Else get left subtree
02EE 686 50$: :

```

```

02EE 687 ; Common processing
02EE 688 ;
56 D5 02EE 689 ; Did we find a matching node?
03 12 02F0 690 ; Br if so - all OK
50 D4 02F2 691 ; Else say we failed
04 04 02F4 692 ; And exit with status
02F5 693 55$:
02F5 694 ;
02F5 695 ; Found a match, copy leaf's data to last BTE and
02F5 696 ; delete the leaf.
02F5 697 ;
Inputs: R9 BTE address of leaf to delete
R6 BTE address of node that matches
02F5 699 ;
53 OC A6 D0 02F5 700 MOVL BTE$B_BTEPTR(R6),R3 ; Save pointer to alternate BTE
OC 53 08 A9 OC 3F BB 02F9 701 PUSHR #*M<R0,R1,R2,R3,R4,R5> ; Save registers
OC A6 OC A9 53 28 A3 02FB 702 SUBW3 #BTE$B_BTEPTR,BTE$B_SIZE(R9),R3 ; Compute size of DATA to move
3F BA 0300 703 MOVW3 R3,BTE$B_BTEPTR(R9),BTE$B_BTEPTR(R6) ; Copy the BTE data
0306 704 POPR #*M<R0,R1,R2,R3,R4,R5> ; Save registers
0308 705 ;
0308 706 ; Now we must go to the other BTE to update it's pointer to
0308 707 ; the new BTE we just reset.
0308 708 ;
OC A9 53 D0 0308 709 MOVL R3,BTE$B_BTEPTR(R9) ; Point other's BTE at new leaf
54 OC A6 D0 030C 710 MOVL BTE$B_BTEPTR(R6),R4 ; Get address of other BTE
29 13 0310 711 BEQL 58$ ; Br if end of recursion
OC A4 56 D0 0312 712 MOVL R6,BTE$B_BTEPTR(R4) ; Update pointer to moved BTE
53 D5 0316 713 TSTL R3 ; All done?
21 13 0318 714 BEQL 58$ ; Br if yes, stop recursing
OC A3 D4 031A 715 CLRL BTE$B_BTEPTR(R3) ; Clear the return pointer
031D 716 ;
031D 717 ; Now we will delete the other BTE.
031D 718 ;
7E 15 A3 9E 031D 719 56$: MOVAB BTE$B_DATA+1(R3),-(SP) ; Push address of data portion of BTE
7E 14 A3 9A 0321 720 MOVZBL BTE$B_DATA(R3),-(SP) ; Push length of data portion of BTE
00000000'EF DF 0325 721 PUSHAL NET$GC_NAME_TREE ; Push address of root of name tree
07 0A A3 E8 032B 722 BLBS BTE$B_TYPE(R3),57$ ; Br if name element
6E 00000000'EF DE 032F 723 MOVAL NET$GC_COL_TREE,(SP) ; Else, set address of other root
FF5F CF 03 FB 0336 724 57$: CALLS #3,DELETE_BTE ; Delete the other BTE first
50 59 D0 033B 725 58$: MOVL R9,R0 ; Address of BTE to delete
00000000'EF 16 033E 726 JSB NET$DEALLOCATE ; Delete the BTE
52 01 CE 0344 727 MNEGL #1,R2 ; Set height change
05 0347 728 RSB ; And return with New decendent in R1
0348 729 60$:
0348 730 ;
0348 731 ; Traverse right subtree
59 DD 0348 732 PUSHL R9 ; Save current BTE
59 50 D0 034A 733 MOVL R0,R9 ; Get subtree pointer
FF64 30 034D 734 BSBW DELETE ; Make recursive call (finds leaf)
59 8ED0 0350 735 POPL R9 ; Get current BTE back
04 A9 51 D0 0353 736 MOVL R1,BTE$B_RIGHT(P9) ; Else, set new subtree ptr
08 A9 52 80 0357 737 ADDB R2,BTE$B_BAL(R9) ; Adjust balance
52 95 035B 738 TSTB R2 ; Any height change?
03 13 035D 739 BEQL 70$ ; Br if not
51 007C 31 035F 740 BRW DEL_REBAL_R ; Else, rebalance if necessary
59 D0 0362 741 70$: MOVL R9,R1 ; Set new parent (leave R2=0)
05 0365 742 RSB ; And exit
0366 743 ;

```

```

0366 745 .SBTTL DEL_REBAL_L - Rebalance after left deletion
0366 746 :+
0366 747 : DEL_REBAL_L - Rebalance after left deletion
0366 748 :
0366 749 : Check the balance of the node. If it is out of balance, rebalance
0366 750 : it with its right decendent
0366 751 :
0366 752 : Inputs: R9 Address of .jtree to be rebalanced
0366 753 :
0366 754 : Outputs: R2 Change in height of subtree (0 or -1)
0366 755 : R1 New parent node of subtree
0366 756 :
0366 757 : R6 destroyed
0366 758 :-
0366 759
0366 760 DEL_REBAL_L:
01 0B A9 91 0366 761 CMPB BTE$B_BAL(R9),#1 : Node gone overweight to the right?
52 0B A9 0C 14 036A 762 BGTR 10$ : Br if so
01 83 036C 763 SUBB3 #1,BTE$B_BAL(R9),R2 : Else set height change according to
52 52 98 0371 764 : balance state
51 59 D0 0371 765 CVTBL R2,R2 : Produce longword result
05 0374 766 MOVL R9,R1 : Set new root node
0377 767 RSB : And exit
0378 768
56 04 A9 D0 0378 769 10$: MOVL BTE$L_RIGHT(R9),R6 : Get right subtree
0B A6 95 037C 770 TSTB BTE$B_BAL(R6) : Test its balance state
28 19 037F 771 BLSS DRL_L : Subtree is left-heavy
14 13 0381 772 BEQL DRL_B : Subtree is balanced
0383 773 :: BGTR DRL_R : Subtree is right-heavy
0383 774
0383 775 :+
0383 776 : Rebalance right-heavy node with right-heavy decendent. Make decendent into
0383 777 : parent (single rotation)
0383 778 :-
0383 779 DRL_R:
04 A9 66 D0 0383 780 MOVL BTE$L_LEFT(R6),BTE$L_RIGHT(R9) ; Make R-son's L-son into R-son
66 59 D0 0387 781 MOVL R9,BTE$L_LEFT(R6) ; make node into R-son's L-son
51 56 D0 038A 782 MOVL R6,R1 ; L-son is new parent
0B A6 94 038D 783 CLRB BTE$B_BAL(R6) ; Both nodes are now balanced
0B A9 94 0390 784 CLRB BTE$B_BAL(R9)
52 01 CE 0393 785 MNEGL #1,R2 ; Height has decreased
05 0396 786 RSB ; Done
0397 787
0397 788 :+
0397 789 : Rebalance right-heavy node with balanced son. Single rotation as above, but
0397 790 : both nodes are unbalanced and there is no overall height change.
0397 791 :-
0397 792 DRL_B:
04 A9 66 D0 0397 793 MOVL BTE$L_LEFT(R6),BTE$L_RIGHT(R9) ; Make R-son's L-son into R-son
66 59 D0 039B 794 MOVL R9,BTE$L_LEFT(R6) ; make node into R-son's L-son
51 56 D0 039E 795 MOVL R6,R1 ; L-son is new parent
0B A9 01 90 03A1 796 MOVB #1,BTE$B_BAL(R9) ; Node is now right unbalanced
0B A6 01 8E 03A5 797 MNEGB #1,BTE$B_BAL(R6) ; New parent is left unbalanced
D4 03A9 798 CLRL R2 ; No height change
05 03AB 799 RSB ; Done
03AC 800
03AC 801 :+

```

```

      03AC 802 : Rebalance right-heavy node with left heavy son. Do double rotation, in
      03AC 803 : which R-son's L-son becomes new parent.
      03AC 804 :-
      03AC 805 DRL_L:
04 50 66 D0 03AC 806 MOVL BTESL_LEFT(R6),R0 ; get 'middle son'
A9 60 D0 03AF 807 MOVL BTESL_LEFT(R0),BTESL_RIGHT(R9) ; make its L-son be new R-son
66 04 A0 D0 03B3 808 MOVL BTESL_RIGHT(R0),BTESL_LEFT(R6) ; give R-son to existing son
60 59 D0 03B7 809 MOVL R9,BTESL_LEFT(R0) ; Make old parent new L-son
04 A0 56 D0 03BA 810 MOVL R6,BTESL_RIGHT(R0) ; And old R-son be new R-son
      OB A9 94 03BE 811 CLRB BTESB_BAL(R9) ; Assume new parent is not R-heavy
      OB A0 95 03C1 812 TSTB BTESB_BAL(R0) ; Was it R-heavy?
      03 15 03C4 813 BLEQ 10$ ; Br if not - all OK
      OB A9 97 03C6 814 DECB BTESB_BAL(R9) ; Else, old parent is L-heavy
      OB A6 94 03C9 815 10$: CLRB BTESB_BAL(R6) ; Assume new parent is not L-heavy
      OB A0 95 03CC 816 TSTB BTESB_BAL(R0) ; Was it?
      03 18 03CF 817 BGEQ 20$ ; Br if not - all OK
      OB A6 96 03D1 818 INCB BTESB_BAL(R6) ; Else, old R-son is R-heavy
      OB A0 94 03D4 819 20$: CLRB BTESB_BAL(R0) ; New parent is balanced
51 50 D0 03D7 820 MOVL R0,R1 ; Set new parent
52 01 CE 03DA 821 MNEGL #1,R2 ; Indicate height change
      05 03DD 822 RSB ; All done
  
```

```

03DE 824      .SBTTL DEL_REBAL_R - Rebalance after right deletion
03DE 825      :+
03DE 826      : DEL_REBAL_R - Rebalance after right deletion
03DE 827      :
03DE 828      : Check the balance of the node. If it is out of balance, rebalance
03DE 829      : it with its left son
03DE 830      :
03DE 831      : Inputs:      R9      Node to be rebalanced
03DE 832      :
03DE 833      : Outputs:     R2      Change in height of subtree (0 or -1)
03DE 834      :              R1      New parent node of subtree
03DE 835      :
03DE 836      :              R6      destroyed
03DE 837      :-
03DE 838      :
03DE 839      DEL_REBAL_R:
FF 8F 0B A9 91 03DE 840      CMPB      BTESB_BAL(R9),#-1      ; Node gone overweight to the left?
03DE 841      BLSS      10$      ; Br if so
52 FF 8F 0B A9 83 03E5 842      SUBB3     BTESB_BAL(R9),#-1,R2      ; Else set height change according to
03EB 843      ; balance state
52 52 98 03EB 844      CVTBL     R2,R2      ; Produce longword result
51 59 D0 03EE 845      MOVL      R9,R1      ; Set new subtree root
05 03F1 846      RSB      ; And return to caller
03F2 847      :
56 69 D0 03F2 848      10$:      MOVL      BTESL_LEFT(R9),R6      ; Get left subtree
0B A6 95 03F5 849      TSTB      BTESB_BAL(R6)      ; And look at its balance state
2D 14 03F8 850      BGTR      DLR_R      ; Subtree is right-heavy
15 13 03FA 851      BEQL      DLR_B      ; Subtree is balanced
03FC 852      ::      BLSS      DLR_L      ; Subtree is left-heavy
03FC 853      :
03FC 854      :+
03FC 855      : Rebalance left-heavy node with left-heavy subtree. Make decendent into
03FC 856      : parent (single rotation)
03FC 857      :-
03FC 858      DLR_L:
69 04 A6 D0 03FC 859      MOVL      BTESL_RIGHT(R6),BTESL_LEFT(R9) ; Make L-son's R-son into L-son
04 A6 59 D0 0400 860      MOVL      R9,BTESL_RIGHT(R6)      ; Make node into L-son's R-son
51 56 D0 0404 861      MOVL      R6,R1      ; L-son is new parent
0B A6 94 0407 862      CLRB      BTESB_BAL(R6)      ; Both nodes are now balanced
0B A9 94 040A 863      CLRB      BTESB_BAL(R9)      ;
52 01 CE 040D 864      MNEGL     #1,R2      ; Height has decreased
05 0410 865      RSB      ; Done
0411 866      :
0411 867      :+
0411 868      : Rebalance left-heavy node with balanced subtree. Single rotation as above,
0411 869      : but both nodes are unbalanced and there is no overall height change.
0411 870      :-
0411 871      DLR_B:
69 04 A6 D0 0411 872      MOVL      BTESL_RIGHT(R6),BTESL_LEFT(R9) ; Make L-son's R-son into L-son
04 A6 59 D0 0415 873      MOVL      R9,BTESL_RIGHT(R6)      ; make node into L-son's R-son
51 56 D0 0419 874      MOVL      R6,R1      ; L-son is new parent
0B A9 01 8E 041C 875      MNEGB     #1,BTESB_BAL(R9)      ; Node is now left unbalanced
0B A6 01 90 0420 876      MOVB     #1,BTESB_BAL(R6)      ; New parent is right unbalanced
52 52 D4 0424 877      CLRL     R2      ; No height change
05 0426 878      RSB      ;
0427 879      :
0427 880      :+
  
```

```

      0427 881 : Rebalance left-heavy node with right heavy subtree. Do double rotation, in
      0427 882 : which L-son's R-son becomes new parent.
      0427 883 :-
      0427 884 DLR_R:
50 04 A6 D0 0427 885      MOVL      BTESL_RIGHT(R6),R0      : Get 'middle son'
69 04 A0 D0 042B 886      MOVL      BTESL_RIGHT(R0),BTESL_LEFT(R9) ; Make its R-son be new L-son
04 A6 60 D0 042F 887      MOVL      BTESL_LEFT(R0),BTESL_RIGHT(R6) ; Give L-son to existing son
04 A0 59 D0 0433 888      MOVL      R9,BTESL_RIGHT(R0)      : Make old parent new R-son
      60 56 D0 0437 889      MOVL      R6,BTESL_LEFT(R0)      : And old L-son be new L-son
      OB A9 94 043A 890      CLRB      BTESB_BAL(R9)      : Assume new parent was not L-heavy
      OB A0 95 043D 891      TSTB      BTESB_BAL(R0)      : Was it L-heavy?
      OB 03 18 0440 892      BGEQ     10$      : Br if not - all OK
      OB A9 96 0442 893      INCB     BTESB_BAL(R9)      : Else, old parent is R-heavy
      OB A6 94 0445 894 10$: CLRB      BTESB_BAL(R6)      : Assume new parent was not R-heavy
      OB A0 95 0448 895      TSTB      BTESB_BAL(R0)      : Was it R-heavy?
      OB 03 15 044B 896      BLEQ     20$      : Br if not - all OK
      OB A6 97 044D 897      DECB     BTESB_BAL(R6)      : Else, old L-son is L-heavy
      OB A0 94 0450 898 20$: CLRB      BTESB_BAL(R0)      : New parent is balanced
51 50 D0 0453 899      MOVL      R0,R1      : Set new parent
52 01 CE 0456 900      MNEGL     #1,R2      : Indicate height change
      05 0459 901      RSB      : All done
  
```

```

045A 903      .SBTTL COMPARE_COLLATE - COMPARE BTE DATA ENTRIES BY COLLATING VALUE
045A 904      .SBTTL COMPARE_COLLATE1 - COMPARE BTE DATA ENTRIES BY COLLATING VALUE
045A 905      :++
045A 906      :
045A 907      : COMPARE THE VALUES OF THE BTE AGAINST THE INPUT KEY
045A 908      :
045A 909      : INPUTS:
045A 910      :
045A 911      : R9 = ADDRESS OF CURRENT NODE, IF COMPARE_COLLATE
045A 912      : R2 = ADDRESS OF CURRENT NODE, IF COMPARE_COLLATE1
045A 913      : R7,R8 = DESCRIPTOR OF STRING TO COMPARE
045A 914      :
045A 915      : OUTPUTS:
045A 916      :
045A 917      : PSL CONDITIONS SET.
045A 918      :--
045A 919      :
045A 920      COMPARE_COLLATE:
15 A9 68 91 045A 921      CMPB      (R8),BTEST_DATA+1(R9)      ; Check 1st char
OF 12 045E 922      BNEQ      90$                          ; Br if thats enough
3E BB 0460 923      PUSHR     #^M<R1,R2,R3,R4,R5>          ; Save registers
51 14 A9 9A 0462 924      MOVZBL   BTEST_DATA(R9),R1        ; Get length of current name
00 68 57 2D 0466 925      CMPC5    R7,(R8),#0,-            ; Compare strings
15 A9 51 046A 926      R1,BTEST_DATA+1(R9)
3E BA 046D 927      POPR      #^M<R1,R2,R3,R4,R5>          ; Restore registers
05 046F 928 90$:    RSB
0470 929
0470 930
0470 931      COMPARE_COLLATE1:
15 A2 68 91 0470 932      CMPB      (R8),BTEST_DATA+1(R2)      ; Check 1st char
OF 12 0474 933      BNEQ      90$                          ; Br if thats enough
3F BB 0476 934      PUSHR     #^M<R0,R1,R2,R3,R4,R5>          ; Save registers
51 14 A2 9A 0478 935      MOVZBL   BTEST_DATA(R2),R1        ; Get length of current name
00 68 57 2D 047C 936      CMPC5    R7,(R8),#0,-            ; Compare strings
15 A2 51 0480 937      R1,BTEST_DATA+1(R2)
3F BA 0483 938      POPR      #^M<R0,R1,R2,R3,R4,R5>          ; Restore registers
05 0485 939 90$:    RSB
0486 940
0486 941
0486 942      .END
  
```


NETTREE
Symbol table

- Subroutines for processing BINARY TRE N² 16-SEP-1984 01:28:51
5-SEP-1984 02:21:41

VAX/VMS Macro V04-00
[NETACP.SRC]NETTREE.MAR;1

Page 22
(15)

NET
V04

ADD NEW BTE	00000176	R	03
BTESB_BAL	0000000B	G	
BTESB_TYPE	0000000A	G	
BTESC_DATA_SIZE	= 00000013		
BTESC_LENGTH	00000027	G	
BTESL_BTEPTR	0000000C	G	
BTESL_LEFT	00000000	G	
BTESL_PTR	00000010	G	
BTESL_RIGHT	00000004	G	
BTEST_DATA	00000014	G	
BTESW_SIZE	00000008	G	
BUGS_NETNOBUF	*****	X	03
BUGS_NETNOSTATE	*****	X	03
BUG_OUT	00000172	R	03
CNFSGET_FIELD	*****	X	03
CNFSL_COLBTE	= 00000000		
CNFSL_NAMEBTE	= 00000004		
CNRSL_COLBTE	= 00000000		
COMPARE_COLLATE	0000045A	R	03
COMPARE_COLLATE1	00000470	R	03
DELETE	000002B4	R	03
DELETE BTE	0000029A	R	03
DEL_REBAL_L	00000366	R	03
DEL_REBAL_R	000003DE	R	03
DLR_B	00000411	R	03
DLR_L	000003FC	R	03
DLR_R	00000427	R	03
DRL_B	00000397	R	03
DRL_L	000003AC	R	03
DRL_R	00000383	R	03
FIND	000000D5	R	03
INSERT	00000196	R	03
NAME_BUF	00000000	R	02
NET\$ADD_NDI	000000FA	RG	03
NET\$ALLOCATE	*****	X	03
NET\$DEALLOCATE	*****	X	03
NET\$DELETE_BTE	00000276	RG	03
NET\$FIND_COL	00000077	RG	03
NET\$FIND_NAME	000000A7	RG	03
NET\$FIND_NDI	00000077	RG	03
NET\$GL_COL_TREE	*****	X	03
NET\$GL_NAME_TREE	*****	X	03
NET\$RESUME_NDI	00000036	RG	03
NET\$TRAVERSE_ALT	0000007A	RG	03
NET\$TRAVERSE_NDI	00000000	RG	03
NFBSC_NDI_COL	= 02020040		
NFBSC_NDI_NNA	= 02020043		
RESUME	00000042	R	03
RESUME_CONT	0000001E	R	03
TRAVERSE	00000013	R	03
SS	= 00000000		

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
. ABS .	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$AB\$\$	00000027 (39.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
NET_IMPURE	00000004 (4.)	02 (2.)	NOPIC USR CON REL LCL NOSHR NOEXE RD WRT NOVEC BYTE
NET_CODE1	00000486 (1158.)	03 (3.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC BYTE

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	22	00:00:00.08	00:00:00.60
Command processing	143	00:00:01.23	00:00:06.48
Pass 1	294	00:00:08.47	00:00:16.08
Symbol table sort	0	00:00:00.80	00:00:01.66
Pass 2	170	00:00:02.46	00:00:03.82
Symbol table output	7	00:00:00.06	00:00:00.09
Psect synopsis output	2	00:00:00.03	00:00:00.03
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	640	00:00:13.13	00:00:28.76

The working set limit was 1950 pages.
44893 bytes (88 pages) of virtual memory were used to buffer the intermediate code.
There were 40 pages of symbol table space allocated to hold 631 non-local and 49 local symbols.
942 source lines were read in Pass 1, producing 19 object records in Pass 2.
18 pages of virtual memory were used to define 17 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
-\$255\$DUA28:[SHRLIB]NMALIBRY.MLB;1	0
-\$255\$DUA28:[SHRLIB]EVCDEF.MLB;1	0
-\$255\$DUA28:[NETACP.OBJ]NETDRV.MLB;1	0
-\$255\$DUA28:[NETACP.OBJ]NET.MLB;1	8
-\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	1
-\$255\$DUA28:[SYSLIB]STARLET.MLB;2	4
TOTALS (all libraries)	13

697 GETS were required to define 13 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LIS\$:NETTREE/OBJ-OBJ\$:NETTREE MSRCS\$:NETTREE/UPDATE=(ENH\$:NETTREE)+EXECMLS/LIB+LIB\$:NET/LIB+LIB\$:NETDRV/LIB+SHRLIB\$:EVCDEF/

