

```
LLL      BBBB BBBB BBBB      RRRRRRRRRRRR
LLL      BBBB BBBB BBBB      RRRRRRRRRRRR
LLL      BBBB BBBB BBBB      RRRRRRRRRRRR
LLL      BBB      BBB      RR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBBB BBBB BBBB      RRRRRRRRRRRR
LLL      BBBB BBBB BBBB      RRRRRRRRRRRR
LLL      BBBB BBBB BBBB      RRRRRRRRRRRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBB      BBB      RRR      RRR
LLL      BBBB BBBB BBBB      RRR      RRR
LLL      BBBB BBBB BBBB      RRR      RRR
LLL      BBBB BBBB BBBB      RRR      RRR
```

IIIIII	NN	NN	DDDDDDDD	EEEEEEEEEE	XX	XX	
IIIIII	NN	NN	DDDDDDDD	EEEEEEEEEE	XX	XX	
II	NN	NN	DD	DD	EE	XX	XX
II	NN	NN	DD	DD	EE	XX	XX
II	NNNN	NN	DD	DD	EE	XX	XX
II	NNNN	NN	DD	DD	EE	XX	XX
II	NN	NN	DD	DD	EEEEEEEE	XX	XX
II	NN	NN	DD	DD	EEEEEEEE	XX	XX
II	NN	NNNN	DD	DD	EE	XX	XX
II	NN	NNNN	DD	DD	EE	XX	XX
II	NN	NN	DD	DD	EE	XX	XX
II	NN	NN	DD	DD	EE	XX	XX
IIIIII	NN	NN	DDDDDDDD	EEEEEEEEEE	XX	XX	
IIIIII	NN	NN	DDDDDDDD	EEEEEEEEEE	XX	XX	

LL	IIIIII	SSSSSSSS
LL	IIIIII	SSSSSSSS
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
LLLLLLLLLL	IIIIII	SSSSSSSS
LLLLLLLLLL	IIIIII	SSSSSSSS

```
0001 0 MODULE LBR_INDEX
0002 0 (IDENT = 'V04-000') = ! Index manipulation routines
0003 1 BEGIN
0004 1
0005 1
0006 1 *****
0007 1 *
0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0010 1 * ALL RIGHTS RESERVED.
0011 1 *
0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0017 1 * TRANSFERRED.
0018 1 *
0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0021 1 * CORPORATION.
0022 1 *
0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0025 1 *
0026 1 *
0027 1 *****
0028 1
0029 1 ++
0030 1
0031 1 FACILITY: Library access procedures
0032 1
0033 1 ABSTRACT:
0034 1
0035 1 The VAX/VMS librarian procedures implement a standard access method
0036 1 to libraries through a shared, common procedure set.
0037 1
0038 1 ENVIRONMENT:
0039 1
0040 1 VAX native, user mode.
0041 1
0042 1 --
0043 1
0044 1
0045 1 AUTHOR: Tim Halvorsen, Benn Schreiber 11-Jun-1979
0046 1
0047 1 MODIFIED BY:
0048 1
0049 1 V03-004 GJA0078 Greg Awdziejewicz 22-Mar-1984
0050 1 put traverse_keys fix back in.
0051 1
0052 1 V03-003 JWi0093 Jim Teague 01-Feb-1983
0053 1 Undo last fix.
0054 1
0055 1 V03-002 JWi0091 Jim Teague 20-Jan-1983
0056 1 Propagate status returned from traverse or traverse2.
0057 1
```

LBR_INDEX
V04=000

F 15
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 BLISS-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 2 (1)

: 58
: 59
: 60
: 61
0058 1 :
0059 1 :
0060 1 :
0061 1 :--

V03-001 JWT0058 Jim Teague 19-Oct-1982
Fix variable-length index module deletion bug.

LBI
VO

Declarations

```

: 63 0062 1 %SBTTL 'Declarations';
: 64 0063 1 LIBRARY 'SYSSLIBRARY:STARLET.L32';      ! VAX/VMS common definitions
: 65 0064 1
: 66 0065 1 REQUIRE 'PREFIX';                      ! Librarian general definitions
: 67 0204 1
: 68 0205 1 REQUIRE 'LBRDEF';                      ! Librarian structure definitions
: 69 0796 1
: 70 0797 1 REQUIRE 'OLDFMTDEF';                   ! Old library format definitions
: 71 0893 1
: 72 0894 1 LINKAGE
: 73 0895 1     fmg_match = JSB (REGISTER = 2, REGISTER = 3,
: 74 0896 1     REGISTER = 4, REGISTER = 5) : NOTUSED (10, 11); ! Linkage for FMG$MATCH_NAME
: 75 0897 1
: 76 0898 1 FORWARD ROUTINE
: 77 0899 1     lbr$set_index,                      ! Set current index number
: 78 0900 1     lbr$lookup_key,                    ! Lookup a key and return RFA
: 79 0901 1     lbr$insert_key,                    ! Insert a key
: 80 0902 1     lbr$replace_key,                  ! Replace rfa for key and modify module header ref. counts
: 81 0903 1     lbr$delete_key,                   ! Delete a key
: 82 0904 1     lbr$get_index,                    ! Return all entries of an index
: 83 0905 1     lbr$search,                      ! Search for all keys assoc. with RFA
: 84 0906 1     check_wild,                      ! Check wildcard name against entry
: 85 0907 1     call_user,                      ! Call user action routine
: 86 0908 1     add_key,                        ! Add a key to a specified index
: 87 0909 1     delete_key,                     ! Delete key from current primary index
: 88 0910 1     remove_key,                     ! Remove a key from a specified index
: 89 0911 1     lookup_key,                     ! Lookup a key and return an RFA
: 90 0912 1     traverse_keys,                  ! Traverse an index one key at a time
: 91 0913 1     create_index,                   ! Create an index block
: 92 0914 1     delete_index,                   ! Deallocate an index block
: 93 0915 1     find_key,                      ! Find key in index structure
: 94 0916 1     key_search,                    ! Binary key search
: 95 0917 1     key_search2,                   ! Variable length keyword search
: 96 0918 1     find_index : JSB_2,             ! Locate index block
: 97 0919 1     add_index,                     ! Add index pointer to parent block
: 98 0920 1     add_index2,                    ! Add index pointer to parent block of variable index
: 99 0921 1     reset_highest,                  ! Reset highest keys in parent blocks
: 100 0922 1     reset_highest2,                ! Reset highest keys in variable len index
: 101 0923 1     parent blocks
: 102 0924 1     check_lock : JSB_0,             ! Check if index is locked from modification
: 103 0925 1     mark_dirty : JSB_1;            ! Mark index block modified
: 104 0926 1
: 105 0927 1 EXTERNAL ROUTINE
: 106 0928 1     fmg$match_name : fmg_match,      ! Perform embedded wild-card matching
: 107 0929 1     make_upper_case : JSB_3,        ! Convert name to upper case, check length
: 108 0930 1     move_to_upper_case : JSB_3,      ! Convert
: 109 0931 1     incr_refcnt,                   ! Increment module reference count
: 110 0932 1     decr_refcnt,                   ! Decrement module reference count
: 111 0933 1     lbr_old_lkp_key,                ! Lookup key in old library
: 112 0934 1     lbr_old_get_idx,               ! Return contents of old library index
: 113 0935 1     lbr_old_src_idx,               ! Search old library index for RFA
: 114 0936 1     read_old_record : JSB_2,        ! Read record from old format library
: 115 0937 1     get_mem : JSB_2,               ! Allocate dynamic memory
: 116 0938 1     get_zmem : JSB_2,              ! Allocate zeroed dynamic memory
: 117 0939 1     dealloc_mem : JSB_2,           ! Deallocate dynamic memory
: 118 0940 1     alloc_block : JSB_2,           ! Allocate disk block
: 119 0941 1     dealloc_block : JSB_1,          ! Deallocate disk block
```

Declarations

```
: 120      0942 1      read_block : JSB_2,      ! Read disk block
: 121      0943 1      read_n_block : JSB_2,    ! Read and cache multiple disk blocks
: 122      0944 1      find_block : JSB_3,      ! Locate disk block and cache it
: 123      0945 1      read_record : JSB_2,     ! Read data record
: 124      0946 1      write_record,           ! Write data record
: 125      0947 1      add_cache : JSB_2,       ! Add cache entry
: 126      0948 1      lookup_cache : JSB_2,    ! Lookup cache entry
: 127      0949 1      empty_cache,            ! Empty cache - write all dirty blocks
: 128      0950 1      set_module,              ! Read module header
: 129      0951 1      incr_rfa : JSB_2,        ! Increment an RFA
: 130      0952 1      validate_ctl : JSB_1;    ! Validate control table index
: 131      0953 1
: 132      0954 1      EXTERNAL
: 133      0955 1          lbr$gl_maxread,      ! Max number of blocks to read at once
: 134      0956 1          lbr$gl_maxidxrd,     ! Max number of blocks in one index read
: 135      0957 1          lbr$gl_control: REF BBLOCK; ! Address of control block
: 136      0958 1
: 137      0959 1      EXTERNAL LITERAL
: 138      0960 1          lbr$_dupkey,
: 139      0961 1          lbr$_illctl,
: 140      0962 1          lbr$_illidxnum,
: 141      0963 1          lbr$_illop,
: 142      0964 1          lbr$_intrnlerr,
: 143      0965 1          lbr$_invkey,
: 144      0966 1          lbr$_invrfa,
: 145      0967 1          lbr$_keynotfnd,
: 146      0968 1          lbr$_libnotopn,
: 147      0969 1          lbr$_nomtchfou,
: 148      0970 1          lbr$_nulidx,
: 149      0971 1          lbr$_updurtrav;
: 150      0972 1
: 151      0973 1
```


LBR\$SET_INDEX

```
0974 1 %SBTTL 'LBR$SET_INDEX';
0975 1 GLOBAL ROUTINE lbr$set_index (ctl_index, index) =
0976 1
0977 1 ---
0978 1
0979 1     Set the current primary index for later operations.
0980 1
0981 1     Inputs:
0982 1
0983 1         ctl_index = Address of longword containing control table index.
0984 1         index = Primary index number
0985 1
0986 1     Outputs:
0987 1
0988 1         lbr$_illidxnum - illegal index number
0989 1         lbr$_libnotopn - library file not open
0990 1         lbr$_insvirmem - insufficient virtual memory
0991 1         lbr$_illctl - illegal control table index
0992 1 ---
0993 1
0994 2 BEGIN
0995 2
0996 2 BUILTIN
0997 2     NULLPARAMETER;                ! True if argument unspecified
0998 2
0999 2
1000 2 perform (validate_ctl (..ctl_index)); ! Validate control table index
1001 2
1002 3 BEGIN
1003 3     BIND
1004 3         header = .lbr$gl_control [lbr$_hdrptr]: BBLOCK; ! Get address of library header
1005 3
1006 3     IF NULLPARAMETER(2)                ! If index number not supplied
1007 3     OR ..index GTRU .header [lhd$_nindex] ! If greater than maximum,
1008 3     OR ..index EQL 0
1009 3     THEN
1010 3         RETURN lbr$_illidxnum;          ! return with error
1011 3
1012 3     lbr$gl_control [lbr$_curidx] = ..index; ! Save current index number
1013 2     END;
1014 2
1015 2 RETURN true;
1016 2
1017 1 END;
```

```
.TITLE LBR_INDEX
.IDENT \V04-000\
```

```
.EXTRN FMG$MATCH_NAME, MAKE_UPPER_CASE
.EXTRN MOVETO_UPPER_CASE
.EXTRN INCR_REFCNT, DECR_REFCNT
.EXTRN LBR_OLD_LKP_KEY
.EXTRN LBR_OLD_GET_IDX
.EXTRN LBR_OLD_SRC_IDX
.EXTRN READ_OLD_RECORD
.EXTRN GET_MEM, GET_ZMEM
```

.EXTRN DEALLOC_MEM, ALLOC_BLOCK
.EXTRN DEALLOC_BLOCK, READ_BLOCK
.EXTRN READ_N_BLOCK, FIND_BLOCK
.EXTRN READ_RECORD, WRITE_RECORD
.EXTRN ADD_CACHE, LOOKUP_CACHE
.EXTRN EMPTY_CACHE, SET_MODULE
.EXTRN INCR_RFA, VALIDATE_CTL
.EXTRN LBR\$GL_MAXREAD, LBR\$GL_MAXIDXRD
.EXTRN LBR\$GL_CONTROL, LBR\$DUPKEY
.EXTRN LBR\$_ILLCTL, LBR\$_ILLIDXNUM
.EXTRN LBR\$_ILLOP, LBR\$_INTRNLERR
.EXTRN LBR\$_INVKEY, LBR\$_INVRFA
.EXTRN LBR\$_KEYNOTFND, LBR\$_LIBNOTOPN
.EXTRN LBR\$_NOMTCHFOU, LBR\$_NULIDX
.EXTRN LBR\$_UPDURTRAV

.PSECT \$CODE\$,NOWRT,2

.ENTRY LBR\$SET_INDEX, Save R2,R3,R4,R5,R6,R7,R8,- : 0975
R9,R10,R11
MOVL @CTL_INDEX, R0 : 1000
BSBW VALIDATE_CTL
BLBC STATUS, 3\$
MOVL LBR\$GL_CONTROL, R1 : 1004
MOVL 10(R1), R0
CMPB (AP), #2 : 1006
BLSSU 1\$
TSTL 8(AP)
BEQL 1\$: 1007
CMPZV #0, #8, 1(R0), @INDEX
BLSSU 1\$: 1008
TSTL @INDEX
BNEQ 2\$: 1010
MOVL #LBR\$_ILLIDXNUM, R0
RET : 1012
MOVL @INDEX, 18(R1) : 1015
MOVL #1, R0 : 1017
RET

OFFC 00000
50 04 BC D0 00002
31 0000G 50 E9 00009
51 0000G CF D0 0000C
50 0A A1 D0 00011
02 6C 91 00015
13 1F 00018
08 AC D5 0001A
0E 13 0001D
08 00 ED 0001F
05 1F 00026
08 BC D5 00028
08 12 0002B
50 00000000G 8F D0 0002D 1\$:
04 00034
12 A1 08 BC D0 00035 2\$:
50 01 D0 0003A
04 0003D 3\$:

; Routine Size: 62 bytes, Routine Base: \$CODE\$ + 0000

LBR\$LOOKUP_KEY

```
198 1018 1 %SBTTL 'LBR$LOOKUP_KEY';
199 1019 1 GLOBAL ROUTINE lbr$lookup_key (ctl_index, key_name, retrfa) =
200 1020 1
201 1021 1 ---
202 1022 1
203 1023 1      Lookup a specified key and return the RFA associated
204 1024 1      with the key.
205 1025 1
206 1026 1      Inputs:
207 1027 1
208 1028 1      ctl_index = Address of a longword containing control table index.
209 1029 1      key_name = Address of descriptor if ASCII keys,
210 1030 1                  or actual binary key.
211 1031 1      retrfa = Address of 6-byte buffer to receive RFA.
212 1032 1
213 1033 1      Outputs:
214 1034 1
215 1035 1      retrfa = RFA associated with key, if found.
216 1036 1
217 1037 1      lbr$_libnotopn - library not open
218 1038 1      lbr$_keynotfnd - key not found
219 1039 1      lbr$_illctl - illegal control table index
220 1040 1
221 1041 1 ---
222 1042 1
223 1043 2 BEGIN
224 1044 2
225 1045 2 MAP
226 1046 2     key_name : REF BBLOCK,           ! Pointer to string descriptor
227 1047 2     retrfa : REF BBLOCK;         ! Pointer to RFA
228 1048 2
229 1049 2 LOCAL
230 1050 2     keydesc : BBLOCK [dsc$_s_bln],
231 1051 2     keynambuf : BBLOCK [lbr$_maxkeylen],
232 1052 2     recdesc : BBLOCK [dsc$_s_bln];
233 1053 2
234 1054 2 BIND
235 1055 2     length = recdesc [dsc$_length] : WORD,
236 1056 2     addr = recdesc [dsc$_pointer] : REF BBLOCK;
237 1057 2
238 1058 2 perform (validate_ctl (..ctl_index)); ! Validate control table index
239 1059 2 keydesc [dsc$_length] = .key_name [dsc$_length]; ! Set length of name
240 1060 2 keydesc [dsc$_pointer] = keynambuf;
241 1061 2 CH$MOVE (.key_name [dsc$_length],
242 1062 2     .key_name [dsc$_pointer], .keydesc [dsc$_pointer]);
243 1063 2
244 1064 3 BEGIN
245 1065 3     BIND
246 1066 3     header = .lbr$_l_control[lbr$_hdrptr] : BBLOCK, ! Pointer to header
247 1067 3     context = .lbr$_l_control[lbr$_ctxptr] : BBLOCK, ! Pointer to context block
248 1068 3     eomodrfa = context[ctx$_eomodrfa] : BBLOCK, ! End of module RFA
249 1069 3     readrfa = context[ctx$_readrfa] : BBLOCK; ! Next RFA for read
250 1070 3
251 1071 3     IF .context[ctx$_oldlib] ! If old format library
252 1072 3     THEN
253 1073 4         BEGIN
254 1074 4             perform(lbr_old_lkp_key (keydesc, .retrfa)); ! Then process elsewhere
```

```

255 1075 4 CH$MOVE(rfa$length, .retrfa, readrfa); ! Set RFA for reading
256 1076 4 CH$FILL(0, rfa$length, eomodrfa); ! Disable end of module
257 1077 4 perform(read_old_record(readrfa, recdesc)); ! Read and skip header
258 1078 4 IF .length NEQ omh$size
259 1079 4 THEN RETURN lbr$_invrfa
260 1080 4 ELSE
261 1081 5 BEGIN
262 1082 5 BIND
263 1083 5 modsizwords = addr[omh$_modsiz] : VECTOR[,WORD];
264 1084 5
265 1085 5 CH$MOVE(rfa$length, .retrfa, eomodrfa);
266 1086 5 incr_rfa(.modsizwords[1] + .modsizwords[0]*%X'10000', eomodrfa);
267 1087 5 END
268 1088 4 END
269 1089 3 ELSE
270 1090 4 BEGIN
271 1091 4 perform (lookup_key (.lbr$_control [lbr$_curidx],
272 1092 4 keydesc, .retrfa));
273 1093 4
274 1094 4 CH$MOVE(rfa$length, .retrfa, readrfa); ! Set for lbr$_get_record
275 1095 4 perform(read_record(readrfa, recdesc)); ! Read module header to skip it
276 1096 4 IF .length NEQ mhd$_mhdlen+.header[lhd$_mhdusz] ! If module header not correct length
277 1097 4 OR .addr[mhd$_refcnt] EQL 0 ! or ref count is 0
278 1098 4 THEN RETURN lbr$_invrfa; ! then RFA is bad
279 1099 3 END;
280 1100 3 context[ctx$_v_lkpdon] = true; ! Indicate lookup_key done
281 1101 2 END;
282 1102 2
283 1103 2 RETURN true;
284 1104 2
285 1105 1 END;
```

				OFFC 00000	.ENTRY	LBR\$LOOKUP_KEY, Save R2,R3,R4,R5,R6,R7,R8,-		
			SE	FF70	CE	9E 00002	R9,R10,R11	1019
			50	04	BC	D0 00007	-144(SP), SP	
					0000G	30 0000B	@CTL_INDEX, R0	1058
			4C		50	E9 0000E	VALIDATE CTL	
			50	08	AC	D0 00011	STATUS, T\$	
		F8	AD		60	B0 00015	KEY_NAME, R0	1059
		FC	AD	08	AE	9E 00019	(R0), KEYDESC	
	FC	BD	04	B0	60	28 0001E	KEYNAMBUF, KEYDESC+4	1060
			52	0000G	CF	D0 00024	(R0), @4(R0), @KEYDESC+4	1062
			57	0A	A2	D0 00029	LBR\$_GL_CONTROL, R2	1066
			56	0E	A2	D0 0002D	10(R2), R7	
			58	0C	AC	D0 00031	14(R2), R6	1067
		4C	04	A6	05	E1 00035	RETRFA, R8	1074
					58	DD 0003A	#5, 4(R6), 2\$	1071
				F8	AD	9F 0003C	R8	1074
			0000G	CF	02	FB 0003F	PUSHL	
			5E		50	E9 00044	PUSHAB	
			68		06	28 00047	KEYDESC	
06	28	A6	00	6E	00	2C 0004C	#2, LBR_OLD_LKP_KEY	
							STATUS, 3\$	
							#6, (R8), 40(R6)	1075
							#0, (SP), #0, #6, 34(R6)	1076

```
; Routine Size: 207 bytes,    Routine Base: $CODES$ + 003E
```

LBR\$INSERT_KEY

```
287 1106 1 %SBTTL 'LBR$INSERT_KEY';
288 1107 1 GLOBAL ROUTINE lbr$insert_key (ctl_index, key_name, rfa) =
289 1108 1
290 1109 1 ---
291 1110 1
292 1111 1     Insert a key into the current primary index.
293 1112 1
294 1113 1     Inputs:
295 1114 1
296 1115 1         ctl_index = Address of control table index.
297 1116 1         key_name = Address of descriptor if ASCII keys,
298 1117 1                   actual key if binary key.
299 1118 1         rfa = Address of RFA to be associated with the key.
300 1119 1
301 1120 1     Outputs:
302 1121 1
303 1122 1         lbr$_libnotopn - library not open
304 1123 1         lbr$_illctl - illegal control table index
305 1124 1         lbr$_dupkey - duplicate key
306 1125 1         lbr$_invrfa - rfa does not point at valid data
307 1126 1 ---
308 1127 1
309 1128 2 BEGIN
310 1129 2
311 1130 2 MAP
312 1131 2     key_name : REF BBLOCK[dsc$_s_bln],
313 1132 2     rfa : REF BBLOCK[rfa$_length];
314 1133 2
315 1134 2 LOCAL
316 1135 2     keydesc : BBLOCK [dsc$_s_bln],
317 1136 2     keynambuf : BBLOCK [lbr$_maxkeylen],
318 1137 2     cachentry : REF BBLOCK;
319 1138 2
320 1139 2 perform (validate_ctl (..ctl_index));    ! Validate control table index
321 1140 2 perform (check_lock ());                ! Verify ability to modify index
322 1141 2 keydesc [dsc$_length] = .key_name [dsc$_length];
323 1142 2 keydesc [dsc$_a_pointer] = keynambuf;
324 1143 2 CH$MOVE (.key_name [dsc$_length],
325 1144 2           .key_name [dsc$_a_pointer], .keydesc [dsc$_a_pointer]);
326 1145 2
327 1146 3 BEGIN
328 1147 3     BIND
329 1148 3         index_desc = .lbr$gl_control[lbr$_hdrptr] + lhd$_idxdesc
330 1149 3         + (.lbr$gl_control[lbr$_curidx]-1)*idd$_length : BBLOCK,
331 1150 3         context = .lbr$gl_control[lbr$_ctxptr] : BBLOCK; ! Context block
332 1151 3
333 1152 3     IF .context[ctx$_oldlib]                ! Cannot insert into old library
334 1153 3     OR .context [ctx$_ronly]                ! or one that is read only
335 1154 3     THEN
336 1155 3         RETURN lbr$_illop;
337 1156 3
338 1157 3     perform (add_key (.lbr$gl_control [lbr$_curidx], keydesc, .rfa));
339 1158 3     perform (incr_refcnt(.rfa));              ! Increment module refernce count
340 1159 3                                           ! updated reference count
341 1160 3     context[ctx$_hdrdirty] = true;          ! Flag header is dirty
342 1161 3     END;
343 1162 2
```

LBR_INDEX
V04=000

LBR\$INSERT_KEY

B 16
15-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 11
(5)

: 344
: 345
: 346
1163 2 RETURN true;
1164 2
1165 1 END;

				OFFC 00000					
		5E	FF78	CE	9E	00002			
		50	04	BC	DO	00007			
				0000G	30	0000B			
		56		50	E9	0000E			
				0000V	30	00011			
		50		50	E9	00014			
		50	08	AC	DO	00017			
		F8	AD	60	BO	0001B			
		FC	AD	6E	9E	0001F			
FC	BD	04	BO	60	28	00023			
		50	0000G	CF	DO	00029			
		52	0E	A0	DO	0002E			
	05	04	A2	05	E0	00032			
			C4	A2	95	00037			
				08	18	0003A			
		50	00000000G	8F	DO	0003C	1\$:		
					04	00043			
			0C	AC	DD	00044	2\$:		
			F8	AD	9F	00047			
			12	A0	DD	0004A			
		0000V	CF	03	FB	0004D			
			12	50	E9	00052			
			0C	AC	DD	00055			
		0000G	CF	01	FB	00058			
			07	50	E9	0005D			
		04	A2	08	88	00060			
			50	01	DO	00064			
				04	00067	3\$:			
.ENTRY LBR\$INSERT_KEY, Save R2,R3,R4,R5,R6,R7,R8,- R9,R10,R11- ; 1107									
MOVAB -136(SP), SP ; 1139									
MOVL @CTL_INDEX, R0 ; 1139									
BSBW VALIDATE_CTL ; 1140									
BLBC STATUS, 3\$; 1140									
BSBW CHECK_LOCK ; 1140									
BLBC STATUS, 3\$; 1141									
MOVL KEY_NAME, R0 ; 1141									
MOVW (R0), KEYDESC ; 1142									
MOVAB KEYNAMBUF, KEYDESC+4 ; 1142									
MOVC3 (R0), @4(R0), @KEYDESC+4 ; 1144									
MOVL LBR\$GL_CONTROL, R0 ; 1148									
MOVL 14(R0), R2 ; 1150									
BBS #5, 4(R2), 1\$; 1152									
TSTB 4(R2) ; 1153									
BGEQ 2\$; 1155									
MOVL #LBR\$_ILLOP, R0 ; 1155									
RET ; 1157									
PUSHL RFA ; 1157									
PUSHAB KEYDESC ; 1158									
PUSHL 18(R0) ; 1158									
CALLS #3, ADD_KEY ; 1158									
BLBC STATUS, 3\$; 1158									
PUSHL RFA ; 1158									
CALLS #1, INCR_REFCNT ; 1160									
BLBC STATUS, 3\$; 1160									
BISB2 #8, 4(R2) ; 1163									
MOVL #1, R0 ; 1163									
RET ; 1165									

; Routine Size: 104 bytes, Routine Base: \$CODE\$ + 0100

LBR\$REPLACE_KEY

```

348 1166 1 %SBTTL 'LBR$REPLACE_KEY';
349 1167 1 GLOBAL ROUTINE lbr$replace_key (ctl_index, key_name, oldrfa, newrfa) =
350 1168 1
351 1169 1 ----
352 1170 1
353 1171 1      Replace the RFA associated with a key with a new rfa.  Update
354 1172 1      the reference counts in both the old and new module headers
355 1173 1
356 1174 1      Inputs:
357 1175 1
358 1176 1          ctl_index = Address of control table index
359 1177 1          key_name = Address of descriptor if ASCII, key if binary
360 1178 1          oldrfa = Address of old rfa
361 1179 1          newrfa = Address of new rfa
362 1180 1
363 1181 1      Outputs:
364 1182 1
365 1183 1          lbr$_libnotopn - library not open
366 1184 1          lbr$_illctl - illegal control table index
367 1185 1          lbr$_invrfa - invalid rfa
368 1186 1
369 1187 1 ----
370 1188 1
371 1189 2 BEGIN
372 1190 2
373 1191 2 MAP
374 1192 2     key_name : REF BBLOCK,
375 1193 2     oldrfa : REF BBLOCK,
376 1194 2     newrfa : REF BBLOCK;
377 1195 2
378 1196 2 LOCAL
379 1197 2     keydesc : BBLOCK [dsc$_s_bln],
380 1198 2     keynambuf : BBLOCK [lbr$_maxkeylen],
381 1199 2
382 1200 2 perform (validate_ctl (..ctl_index));          ! Validate control table index
383 1201 2 keydesc [dsc$_length] = .key_name [dsc$_length];
384 1202 2 keydesc [dsc$_pointer] = keynambuf;
385 1203 2 CH$MOVE (.key_name [dsc$_length],
386 1204 2     .key_name [dsc$_pointer], .keydesc [dsc$_pointer]);
387 1205 2
388 1206 3 BEGIN
389 1207 3     LOCAL
390 1208 3         vbn,
391 1209 3         index_block,
392 1210 3         offset,
393 1211 3         addpos,
394 1212 3         entry : REF BBLOCK;
395 1213 3
396 1214 3     BIND
397 1215 3         context = .lbr$_gl_control [lbr$_l_ctxptr] : BBLOCK;
398 1216 3
399 1217 3     IF .context [ctx$_oldlib]
400 1218 3     OR .context [ctx$_only]
401 1219 3     THEN RETURN lbr$_itlop;
402 1220 3
403 1221 3 ! First make sure its a real key.  If not found, treat as an insert
404 1222 3
```



```

: 405 1223 3 IF NOT find_key (.lbr$gl_control [lbr$curidx], keydesc, 0,
: 406 1224 3 vbn, index_block, offset, addpos)
: 407 1225 3 THEN RETURN lbr$insert_key (.ctl_index, .key_name, .newrfa);
: 408 1226 3 entry = .index_block + index$entries + .offset; ! Point to entry
: 409 1227 3 IF NOT CH$EQL (rfa$length, entry [idx$b_rfa], rfa$length, .oldrfa)
: 410 1228 3 THEN RETURN lbr$_invrfa;
: 411 1229 3
: 412 1230 3 ! Decrement ref. count in old module header
: 413 1231 3 perform (decr_refcnt (.oldrfa));
: 414 1232 3
: 415 1233 3 ! Increment ref. count in new module header
: 416 1234 3 perform (incr_refcnt (.newrfa));
: 417 1235 3
: 418 1236 3
: 419 1237 3 ! Update index entry
: 420 1238 3 CH$MOVE (rfa$length, .newrfa, entry [idx$b_rfa]);
: 421 1239 3 mark_dirty (.vbn); ! Mark index block dirty
: 422 1240 3
: 423 1241 3 END;
: 424 1242 2
: 425 1243 2 RETURN true
: 426 1244 1 END;

```

! Of lbr\$replace_key

				OFFC 00000	.ENTRY	LBR\$REPLACE_KEY, Save R2,R3,R4,R5,R6,R7,R8,-;					
			SE	FF68	CE	9E	00002	MOVAB	R9,R10,R11		1167
			50	04	BC	DO	00007	MOVL	-152(SP), SP		
					0000G	30	0000B	BSBW	@CTL_INDEX, R0		1200
			77		50	E9	0000E	BLBC	VALIDATE CTL		
			56	08	AC	DO	00011	MOVL	STATUS, 5\$		
		F8	AD		66	BO	00015	MOVW	KEY NAME, R6		1201
		FC	AD	10	AE	9E	00019	MOVW	(R6), KEYDESC		
FC	BD	04	B6		66	28	0001E	MOVAB	KEYNAMBUF, KEYDESC+4		1202
			51	0000G	CF	DO	00024	MOVAB	(R6), @4(R6), @KEYDESC+4		1204
			50	0E	A1	DO	00029	MOVL	LBR\$GL_CONTROL, R1		1215
		05	A0		05	E0	0002D	MOVL	14(R1), R0		
				04	A0	95	00032	BBS	#5, 4(R0), 1\$		1217
					08	18	00035	TSTB	4(R0)		1218
			50	00000000G	8F	DO	00037	BGEQ	2\$		
						04	0003E	MOVL	#LBR\$_ILLOP, R0		1219
					5E	DD	0003F	RET			
				08	AE	9F	00041	PUSHL	SP		1223
				10	AE	9F	00044	PUSHAB	OFFSET		
				18	AE	9F	00047	PUSHAB	INDEX_BLOCK		
					7E	D4	0004A	PUSHAB	VBN		
			F8		AD	9F	0004C	CLRL	-(SP)		
			12		A1	DD	0004F	PUSHAB	KEYDESC		
		0000V	CF		07	FB	00052	PUSHL	18(R1)		
			OE		50	E8	00057	CALLS	#7, FIND_KEY		
				10	AC	DD	0005A	BLBS	R0, 3\$		
					56	DD	0005D	PUSHL	NEWRFA		1225
				04	AC	DD	0005F	PUSHL	R6		
		FF31	CF		03	FB	00062	PUSHL	CTL_INDEX		
								CALLS	#3, LBR\$INSERT_KEY		

LBR_INDEX
V04=000

LBR\$REPLACE_KEY

E 16
16-Sep-1984 01:56:12 VAX-11 Bliss-32 V4.0-742 Page 14
14-Sep-1984 12:37:41 DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1 (6)

54	08	AE	04	AE	04 00067	RET		
		54		0C	C1 00068 3\$:	ADDL3	OFFSET, INDEX_BLOCK, R4	: 1226
OC	BC	64		06	C0 0006E	ADDL2	#12, ENTRY	: 1227
				08	29 00071	CMPC3	#6, (ENTRY), @OLDRFA	: 1228
		50	00000000G	8F	13 00076	BEQL	4\$	: 1232
					D0 00078	MOVL	#LBR\$_INVRFA, R0	: 1236
					04 0007F	RET		: 1240
	0000G	CF	0C	AC	DD 00080 4\$:	PUSHL	OLDRFA	: 1241
		1A		01	FB 00083	CALLS	#1, DECR_REFCNT	: 1243
				50	E9 00088 5\$:	BLBC	STATUS, 8\$	: 1244
			10	AC	DD 0008B	PUSHL	NEWFA	: 1240
	0000G	CF		01	FB 0008E	CALLS	#1, INCR_REFCNT	: 1241
		OF		50	E9 00093	BLBC	STATUS, 8\$	: 1243
64	10	BC		06	28 00096	MOVC3	#6, @NEWFA, (ENTRY)	: 1244
		50	0C	AE	D0 0009B	MOVL	VBN, R0	: 1245
					0000V 30 0009F	BSBW	MARK DIRTY	: 1246
		50		01	D0 000A2	MOVL	#1, R0	: 1247
					04 000A5 6\$:	RET		: 1248

; Routine Size: 166 bytes, Routine Base: \$CODE\$ + 0175

LBR\$DELETE_KEY

```
428 1245 1 %SBTTL 'LBR$DELETE_KEY';
429 1246 1 GLOBAL ROUTINE lbr$delete_key (ctl_index, key_name) =
430 1247 1
431 1248 1 |---
432 1249 1 |
433 1250 1 |         Delete a specified key from the current primary index.
434 1251 1 |
435 1252 1 |     Inputs:
436 1253 1 |
437 1254 1 |         ctl_index = Address of control table index.
438 1255 1 |         key_name = Address of string descriptor or binary key.
439 1256 1 |
440 1257 1 |     Outputs:
441 1258 1 |
442 1259 1 |         lbr$_libnotopn - library not open
443 1260 1 |         lbr$_illctl - illegal control table index
444 1261 1 |         lbr$_keynotfnd - key not found
445 1262 1 |---
446 1263 1
447 1264 2 BEGIN
448 1265 2
449 1266 2 MAP
450 1267 2     key_name : REF BBLOCK;
451 1268 2
452 1269 2 LOCAL
453 1270 2     keydesc : BBLOCK [dsc$_s_bln],
454 1271 2     keynambuf : BBLOCK [lbr$_maxkeylen];
455 1272 2
456 1273 2 perform (validate_ctl (..ctl_index));    ! Validate control table index
457 1274 2 perform (check_lock ());                ! Verify ability to modify index
458 1275 2 keydesc [dsc$_length] = .key_name [dsc$_length];
459 1276 2 keydesc [dsc$_pointer] = keynambuf;
460 1277 2 CH$MOVE (.key_name [dsc$_length],
461 1278 2     .key_name [dsc$_pointer], .keydesc [dsc$_pointer]);
462 1279 2
463 1280 2
464 1281 2 perform (delete_key (keydesc));            ! Delete the key
465 1282 2 RETURN true
466 1283 1 END;
```

				OFFC 00000	.ENTRY	LBR\$DELETE_KEY, Save R2,R3,R4,R5,R6,R7,R8,- ;	
		5E	FF78	CE 9E 00002	MOVAB	R9,R10,R11-	1246
		50	04	BC D0 00007	MOVL	-136(SP), SP	
				0000G 30 0000B	BSBW	@CTL_INDEX, R0	1273
		26		50 E9 0000E	BLBC	VALIDATE_CTL	
				0000V 30 00011	BSBW	STATUS, T\$	
		20		50 E9 00014	BLBC	CHECK_LOCK	1274
		50	08	AC D0 00017	MOVL	STATUS, 1\$	
		F8 AD		60 B0 0001B	MOVL	KEY_NAME, R0	1275
		FC AD		6E 9E 0001F	MOVW	(R0), KEYDESC	
FC	BD	04	B0	60 28 00023	MOVAB	KEYNAMBUF, KEYDESC+4	1276
			F8 AD	9F 00029	MOVW	(R0), @4(R0), @KEYDESC+4	1278
					PUSHAB	KEYDESC	1281

LBR_INDEX
V04=000

LBR\$DELETE_KEY

0000V

CF
03
50

01 FB 0002C
50 E9 00031
01 D0 00034
04 00037 1\$:

CALLS #1, DELETE_KEY
BLBC STATUS, 1\$-
MOVL #1, R0
RET

G 16
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 16
(7)

:
:
: 1282
: 1283

; Routine Size: 56 bytes, Routine Base: \$CODE\$ + 021B

delete_key

```

468 1284 1 %SBTTL 'delete_key';
469 1285 1 GLOBAL ROUTINE delete_key (key_name) =
470 1286 1
471 1287 1 ---
472 1288 1
473 1289 1 Delete a key from the current primary index
474 1290 1
475 1291 1 Inputs:
476 1292 1
477 1293 1 key_name = Address of string descriptor or binary key
478 1294 1
479 1295 1 Outputs:
480 1296 1
481 1297 1 ---
482 1298 1
483 1299 2 BEGIN
484 1300 2
485 1301 2 LOCAL
486 1302 2 localrfa : BBLOCK[rfa$length];
487 1303 2
488 1304 2 BIND
489 1305 2 context = .lbr$gl_control[lbr$l_ctxptr] : BBLOCK;
490 1306 2
491 1307 2 IF .context[ctx$v_oldlib] ! Cannot modify old libraries
492 1308 2 OR .context[ctx$v_ronly] ! or read only libraries
493 1309 2 THEN
494 1310 2 RETURN lbr$_illob;
495 1311 2
496 P 1312 2 perform(lookup_key(.lbr$gl_control[lbr$l_curidx],
497 1313 2 .key_name, localrfa));
498 1314 2
499 1315 2 perform(remove_key(.lbr$gl_control[lbr$l_curidx], .key_name));
500 1316 2
501 1317 2 perform(decr_refcnt(localrfa)); !Decrement reference count
502 1318 2
503 1319 2 context[ctx$v_hdrdirty] = true; !Flag header is dirty
504 1320 2
505 1321 2 RETURN true;
506 1322 2
507 1323 1 END;
```

				0004 00000	.ENTRY DELETE_KEY, Save R2	: 1285
	5E		08	C2 00002	SUBL2 #8, SP	: 1305
	50	0000G	CF	D0 00005	MOVL LBR\$GL_CONTROL, R0	: 1307
	52	0E	A0	D0 0000A	MOVL 14(R0), R2	: 1308
05	04	A2	05	E0 0000E	BBS #5, 4(R2), 1\$	: 1310
		04	A2	95 00013	TSTB 4(R2)	: 1313
			08	18 00016	BGEQ 2\$	
	50	00000000G	8F	D0 00018 1\$:	MOVL #LBR\$_ILLOB, R0	
				04 0001F	RET	
			5E	DD 00020 2\$:	PUSHL SP	
		04	AC	DD 00022	PUSHL KEY_NAME	
		12	A0	DD 00025	PUSHL 18(R0)	

LBR_INDEX
V04=000

delete_key

I 16
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 18
(8)

0000V	CF		03	FB	00028	CALLS	#3, LOOKUP_KEY	:	
	24		50	E9	0002D	BLBC	STATUS, 3\$	:	
		04	AC	DD	00030	PUSHL	KEY NAME	:	1315
	50	0000G	CF	D0	00033	MOVL	LBR\$GL_CONTROL, R0	:	
		12	A0	DD	00038	PUSHL	18(R0)	:	
0000V	CF		02	FB	0003B	CALLS	#2, REMOVE_KEY	:	
	11		50	E9	00040	BLBC	STATUS, 3\$	:	
			5E	DD	00043	PUSHL	SP	:	1317
0000G	CF		01	FB	00045	CALLS	#1, DECR_REFCNT	:	
	07		50	E9	0004A	BLBC	STATUS, 3\$	:	
04	A2		08	88	0004D	BISB2	#8, 4(R2)	:	1319
	50		01	D0	00051	MOVL	#1, R0	:	1321
			04	00054	3\$:	RET		:	1323

; Routine Size: 85 bytes, Routine Base: \$CODE\$ + 0253

LBR\$GET_INDEX

```
509 1324 1 %SBTTL 'LBR$GET_INDEX';
510 1325 1 GLOBAL ROUTINE lbr$get_index (ctl_index, index, user_routine, match_desc) =
511 1326 1
512 1327 1 ---
513 1328 1
514 1329 1     Call a user-supplied routine for each key in the specified
515 1330 1     primary index.
516 1331 1
517 1332 1 Inputs:
518 1333 1
519 1334 1     ctl_index = Address of the control table index
520 1335 1     index = Address of the primary index number
521 1336 1     user_routine = Address of user action routine
522 1337 1     match_desc = Address (optional) of string descriptor for matching
523 1338 1
524 1339 1 Outputs:
525 1340 1
526 1341 1     The action routine is called once for each key in the index.
527 1342 1
528 1343 1     lbr$_libnotopn - library not open
529 1344 1     lbr$_illctl - illegal control table index
530 1345 1     lbr$_illidxnum - illegal index number
531 1346 1 ---
532 1347 1
533 1348 2 BEGIN
534 1349 2
535 1350 2 MAP
536 1351 2     match_desc : REF BBLOCK;
537 1352 2
538 1353 2 LOCAL
539 1354 2     keydesc : BBLOCK [dsc$_s_bln],
540 1355 2     keynambuf : BBLOCK [lbr$_maxkeylen],
541 1356 2     wildcard;
542 1357 2
543 1358 2 BUILTIN
544 1359 2     NULLPARAMETER;                ! True if argument unspecified
545 1360 2
546 1361 2 perform (validate_ctl (..ctl_index)); ! Validate control table index
547 1362 2
548 1363 3 BEGIN
549 1364 3     BIND
550 1365 3         header = .lbr$gl_control [lbr$_hdrptr] : BBLOCK, ! Address the library header
551 1366 3         context = .lbr$gl_control[lbr$_ctxptr] : BBLOCK,
552 1367 3         index_desc = .lbr$gl_control[lbr$_hdrptr] + lhd$_idxdesc
553 1368 3             + (.lbr$gl_control[lbr$_cuidx]-1)*idd$_length : BBLOCK;
554 1369 3
555 1370 3     IF ..index GTRU .header [lhd$_nindex] ! If illegal index number,
556 1371 3     OR ..index EQL 0
557 1372 3     THEN
558 1373 3         RETURN lbr$_illidxnum; ! return with error
559 1374 3
560 1375 3     wildcard = false; ! Assume no wildcards
561 1376 3     IF NOT NULLPARAMETER(4) ! If argument is present
562 1377 3         AND .match_desc [dsc$_length] NEQ 0 ! and non-null
563 1378 3         AND .match_desc [dsc$_pointer] NEQ 0 !
564 1379 4     THEN BEGIN ! Then do wild card matching
565 1380 4         wildcard = true; ! then there is a match descriptor
```

```

566      1381  4      keydesc [dsc$w_length] = .match_desc [dsc$w_length];
567      1382  4      keydesc [dsc$a_pointer] = keynambuf;
568      1383  4      IF .index_desc [idd$w_nocasecmp]
569      1384  4      THEN
570      1385  4          CH$MOVE (.match_desc [dsc$w_length],
571      1386  4              .match_desc [dsc$a_pointer], .keydesc [dsc$a_pointer])
572      1387  4      ELSE
573      1388  4          perform (make_upper_case (.match_desc, keydesc, true));
574      1389  3      END;
575      1390  3
576      1391  3      context [ctx$w_found1] = false;          !Flag no matches found
577      1392  3      IF .context[ctx$w_oldlib]                ! If old format library
578      1393  3      THEN perform (lbr_old_get_idx (..index, .user_routine, (IF .wildcard
579      1394  3          THEN .match_desc
580      1395  4          ELSE 0)))
581      1396  4
582      1397  3      ELSE perform (traverse_keys (..index, (IF .wildcard          ! Traverse the index
583      1398  3          THEN check_wild          ! looking for matches
584      1399  3          ELSE call_user), .user_routine, ! or just calling user
585      1400  3          (IF .wildcard
586      1401  3          THEN .match_desc
587      1402  3          ELSE 0)));
588      1403  3      IF NOT .context [ctx$w_found1]                !If no matches found
589      1404  4          THEN RETURN (IF .wildcard
590      1405  4              THEN lbr$_nomtchfou
591      1406  4              ELSE lbr$_nulidx
592      1407  4              )
593      1408  3          ELSE RETURN true;
594      1409  3
595      1410  2      END;
596      1411  2
597      1412  1  END;

```

!Of lbr\$get_index

				OFFC 00000	.ENTRY	LBR\$GET_INDEX, Save R2,R3,R4,R5,R6,R7,R8,-	1325
						R9,R10,R11	
					MOVAB	-136(SP), SP	
					MOVL	@CTL_INDEX, R0	1361
					BSBW	VALIDATE CTL	
					BLBC	STATUS, 4\$	
					MOVL	LBR\$GL_CONTROL, R0	1365
					MOVL	10(R0), R2	
					MOVL	14(R0), R6	1366
					MOVL	18(R0), R1	1368
					MOVAQ	@10(R0)[R1], R1	
					MOVAB	188(R1), R1	
					CMPZV	#0, #8, 1(R2), @INDEX	1370
					BLSSU	1\$	
					TSTL	@INDEX	1371
					BNEQ	2\$	
					MOVL	#LBR\$_ILLIDXNUM, R0	1373
					RET		
					CLRL	WILDCARD	1375
					CMPB	(AP), #4	1376

				3B	1F	00047	BLSSU	5\$	
			10	AC	D5	00049	TSTL	16(AP)	
				36	13	0004C	BEQL	5\$	
			10	BC	B5	0004E	TSTW	@MATCH_DESC	1377
				31	13	00051	BEQL	5\$	
		50	10	AC	D0	00053	MOVL	MATCH_DESC, R0	1378
			04	A0	D5	00057	TSTL	4(R0)	
				28	13	0005A	BEQL	5\$	
		57		01	D0	0005C	MOVL	#1, WILDCARD	1380
		50	10	AC	D0	0005F	MOVL	MATCH_DESC, R0	1381
	F8	AD		60	B0	00063	MOVW	(R0), KEYDESC	
	FC	AD		6E	9E	00067	MOVAB	KEYNAMBUF, KEYDESC+4	1382
		61		03	E1	0006B	BBC	#3, (R1), 3\$	1383
FC	08		04	B0	60	28	MOVC3	(R0), @4(R0), @KEYDESC+4	1386
BD					0D	11	BRB	5\$	1385
		51	F8	AD	9E	00077	MOVAB	KEYDESC, R1	1388
		52		01	D0	0007B	MOVL	#1, R2	
				0000G	30	0007E	BSBW	MAKE UPPER CASE	
		65		50	E9	00081	BLBC	STATUS, 16\$	
		04	40	8F	8A	00084	BICB2	#64, 4(R6)	1391
17		04		05	E1	00089	BBC	#5, 4(R6), 8\$	1392
		05		57	E9	0008E	BLBC	WILDCARD, 6\$	1395
			10	AC	DD	00091	PUSHL	MATCH_DESC	
				02	11	00094	BRB	7\$	
				7E	D4	00096	CLRL	-(SP)	
			0C	AC	DD	00098	PUSHL	USER ROUTINE	
			08	BC	DD	0009B	PUSHL	@INDEX	
		0000G	CF	03	FB	0009E	CALLS	#3, LBR_OLD_GET_IDX	
				26	11	000A3	BRB	13\$	
		05		57	E9	000A5	BLBC	WILDCARD, 9\$	1402
			10	AC	DD	000A8	PUSHL	MATCH_DESC	
				02	11	000AB	BRB	10\$	
				7E	D4	000AD	CLRL	-(SP)	
			0C	AC	DD	000AF	PUSHL	USER ROUTINE	
		07		57	E9	000B2	BLBC	WILDCARD, 11\$	
		50	0000V	CF	9E	000B5	MOVAB	CHECK_WILD, R0	
				05	11	000BA	BRB	12\$	
		50	0000V	CF	9E	000BC	MOVAB	CALL_USER, R0	
				50	DD	000C1	PUSHL	R0	
			08	BC	DD	000C3	PUSHL	@INDEX	
		0000V	CF	04	FB	000C6	CALLS	#4, TRAVERSE_KEYS	
13		04		50	E9	000CB	BLBC	STATUS, 16\$	
				06	E0	000CE	BBS	#6, 4(R6), 15\$	1403
		08		57	E9	000D3	BLBC	WILDCARD, 14\$	1404
		50	00000000G	8F	D0	000D6	MOVL	#LBR\$_NOMTCHFOU, R0	
					04	000DD	RET		
		50	00000000G	8F	D0	000DE	MOVL	#LBR\$_NULIDX, R0	
					04	000E5	RET		1408
		50		01	D0	000E6	MOVL	#1, R0	
					04	000E9	RET		1412

; Routine Size: 234 bytes, Routine Base: \$CODE\$ + 02A8

LBR\$SEARCH

```
599 1413 1 %SBTTL 'LBR$SEARCH';
600 1414 1 GLOBAL ROUTINE lbr$search (ctl_index, index, rfa, user_routine) =
601 1415 1
602 1416 1 ---
603 1417 1
604 1418 1 Search a specified primary index for all keys associated
605 1419 1 with a given RFA. The user supplied action routine will
606 1420 1 be called for each key associated with the RFA.
607 1421 1
608 1422 1 Inputs:
609 1423 1
610 1424 1     ctl_index = Address of the control table index
611 1425 1     index = Address of the primary index number
612 1426 1     rfa = Address of the RFA to be searched for
613 1427 1     user_routine = Address of user supplied action routine.
614 1428 1
615 1429 1 Outputs:
616 1430 1
617 1431 1     The action routine will be called for each key found.
618 1432 1
619 1433 1 ---
620 1434 1
621 1435 2 BEGIN
622 1436 2
623 1437 2 MAP
624 1438 2     rfa: REF BBLOCK;           ! Access as RfA structure
625 1439 2
626 1440 2 ROUTINE check_rfa (entry, user_routine, index_desc, test_rfa) =
627 1441 3 BEGIN
628 1442 3 MAP
629 1443 3     test_rfa : REF BBLOCK[rfa$length],
630 1444 3     index_desc: REF BBLOCK,
631 1445 3     entry: REF BBLOCK;
632 1446 3 IF .entry [idx$length] EQL .test_rfa [rfa$length]
633 1447 3 AND .entry [idx$offset] EQL .test_rfa [rfa$offset]
634 1448 3 THEN
635 1449 3     perform (call_user (.entry, .user_routine, .index_desc));
636 1450 3 RETURN true;
637 1451 2 END;
```

0000 00000 CHECK_RFA:									
	50	04	AC	D0	00002	WORD	Save nothing		1440
	51	10	AC	D0	00006	MOVL	ENTRY, R0		1446
	61		60	D1	0000A	MOVL	TEST_RFA, R1		
			15	12	0000D	CMPL	(R0), (R1)		
						BNEQ	1\$		
04	A1	04	A0	B1	0000F	CMPL	4(R0), 4(R1)		1447
			0E	12	00014	BNEQ	1\$		
	7E	08	AC	7D	00016	MOVQ	USER_ROUTINE, -(SP)		1449
			50	DD	0001A	PUSHL	R0		
0000v	CF		03	FB	0001C	CALLS	#3, CALL_USER		
	03		50	E9	00021	BLBC	STATUS, 2\$		
	50		01	D0	00024	MOVL	#1, R0		1450

04 00027 2\$: RET

; 1451

; Routine Size: 40 bytes, Routine Base: \$CODE\$ + 0392

```
: 638      1452  2
: 639      1453  2
: 640      1454  2 perform (validate_ctl(..ctl_index)); ! Validate control table index
: 641      1455  2
: 642      1456  2 BEGIN
: 643      1457  2 BIND
: 644      1458  2 context = .lbr$gl_control[lbr$l_ctxptr] : BBLOCK; ! Address the context block
: 645      1459  2 header = .lbr$gl_control[lbr$l_hdrptr] : BBLOCK; ! Address the library header
: 646      1460  2
: 647      1461  2 IF ..index GTRU .header[lhd$b_nindex] ! If illegal index number,
: 648      1462  2 OR ..index EQL 0
: 649      1463  2 THEN
: 650      1464  2 RETURN lbr$_illidxnum; ! return with error
: 651      1465  2
: 652      1466  2 IF .context[ctx$v_oldlib] ! If old format library
: 653      1467  2 THEN RETURN lbr$_old_src_idx(..index, .rfa, .user_routine);
: 654      1468  2
: 655      1469  2 perform (traverse_keys(..index, check_rfa, .user_routine, .rfa));
: 656      1470  2 END;
: 657      1471  2
: 658      1472  2 RETURN true;
: 659      1473  2
: 660      1474  1 END;
```

				OFFC 00000	.ENTRY	LBR\$SEARCH, Save R2,R3,R4,R5,R6,R7,R8,R9,-	
			50	04 BC D0 00002	MOVL	R10,R11	: 1414
			48	0000G 30 00006	BSBW	@CTL_INDEX, R0	: 1454
			50	0000G CF D0 0000C	BLBC	VALIDATE_CTL	
			50	0A A0 7D 00011	MOVL	STATUS, 4\$	
08	5C	01	08	00 00 ED 00015	MOVL	LBR\$GL_CONTROL, R0	: 1458
				05 1F 0001C	MOVQ	10(R0), R0	: 1459
				08 BC D5 0001E	CMPZV	#0, #8, 1(R0), @INDEX	: 1461
				08 12 00021	BLSSU	1\$	
				8F D0 00023 1\$:	TSTL	@INDEX	: 1462
				04 0002A	BNEQ	2\$	
			50	00000000G 8F D0 00023 1\$:	MOVL	#LBR\$_ILLIDXNUM, R0	: 1464
				05 E1 0002B 2\$:	RET		
		0D	04	A1 0C AC 7D 00030	BBC	#5, 4(R1), 3\$	: 1466
			7E	08 BC DD 00034	MOVQ	RFA, -(SP)	: 1467
				03 FB 00037	PUSHL	@INDEX	
			0000G	CF 04 0003C	CALLS	#3, LBR_OLD_SRC_IDX	
				0C AC DD 0003D 3\$:	RET		
				10 AC DD 00040	PUSHL	RFA	: 1469
				92 AF 9F 00043	PUSHL	USER_ROUTINE	
				08 BC DD 00046	PUSHAB	CHECK_RFA	
			0000V	CF 04 FB 00049	PUSHL	@INDEX	
			03	50 E9 0004E	CALLS	#4, TRAVERSE_KEYS	
					BLBC	STATUS, 4\$	

LBR_INDEX
V04=000

LBR\$SEARCH

50

01

D0 00051

04 00054 4\$:

MOVL

#1, R0

RET

C 1
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VM\$MASTER:[LBR.SRC]INDEX.B32;1

Page 24
(10)

: 1472

: 1474

; Routine Size: 85 bytes, Routine Base: \$CODE\$ + 03BA

LBR_INDEX
V04=000

check_wild

```

662 1475 1 %SBTTL 'check_wild';
663 1476 1 ROUTINE check_wild (entry, user_routine, index_desc, match_desc) =
664 1477 2 BEGIN
665 1478 2 ---
666 1479 2      Called by traverse for each entry in the index. Check to
667 1480 2      see if current entry matches the match_desc. Call user if so.
668 1481 2
669 1482 2      Inputs:
670 1483 2
671 1484 2      entry = Address of key entry
672 1485 2      user_routine = Address of user action routine
673 1486 2      index_desc = Address of index descriptor for index
674 1487 2      match_desc = string descriptor for match string
675 1488 2
676 1489 2      ---
677 1490 2 MAP
678 1491 2     entry : REF BBLOCK,
679 1492 2     index_desc : REF BBLOCK,
680 1493 2     match_desc : REF BBLOCK;
681 1494 2 LOCAL
682 1495 2     entrykey : BBLOCK [lbr$$_maxkeylen];
683 1496 2
684 1497 2 IF .index_desc [idd$$_upcasntry]
685 1498 2 THEN
686 1499 3     BEGIN
687 1500 3     moveto_upper_case (.entry [idx$$_keylen], entry [idx$$_keyname], entrykey)
688 1501 3     END
689 1502 2 ELSE
690 1503 2     CH$MOVE (.entry [idx$$_keylen], entry [idx$$_keyname], entrykey);
691 1504 2
692 1505 3 IF (NOT .index_desc [idd$$_ascii] ! If not ASCII keys
693 1506 4 OR (fmg$match_name (.entry [idx$$_keylen], entrykey,
694 1507 4     .match_desc [dsc$$_length],
695 1508 4     .match_desc [dsc$$_pointer])
696 1509 4     OR CH$EQL (.match_desc [dsc$$_length], entrykey,
697 1510 4     .match_desc [dsc$$_length],
698 1511 3     .match_desc [dsc$$_pointer])))
699 1512 2     THEN perform (call_user (.entry, .user_routine, .index_desc, .match_desc));
700 1513 2 RETURN true
701 1514 1 END;
!Of check_wild
```

OFFC 00000 CHECK_WILD:

					.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	1476	
	5E	80	AE	9E	00002	MOVAB	-128(SP), SP	
	50	04	AC	00	00006	MOVL	ENTRY, R0	1500
	57	04	AC	00	0000A	MOVL	ENTRY, R7	
10	0C	BC	05	E1	0000E	BBC	#5, @INDEX_DESC, 1\$	1497
	52		6E	9E	00013	MOVAB	ENTRYKEY, R2	1500
	51	07	A0	9E	00016	MOVAB	7(R0), R1	
	50	06	A7	9A	0001A	MOVZBL	6(R7), R0	
			0000G	30	0001E	BSBW	MOVETO_UPPER_CASE	
			09	11	00021	BRB	2\$	1499
	51	06	A7	9A	00023	MOVZBL	6(R7), R1	1503

LBR_INDEX
V04=000

check_wild

E 1
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 BLISS-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 26
(11)

6E

07

A0

0C

51

28

00027

2\$:

MOV C3

R1, 7(R0), ENTRYKEY

BLBC

@INDEX_DESC, 3\$

MOVL

MATCH_DESC, R6

MOVAB

ENTRYKEY, R3

MOVL

4(R6), R5

MOVZWL

(R6), R4

MOVZBL

6(R7), R2

BSBW

FMG\$MATCH_NAME

BLBS

R0, 3\$

CMPC3

(R6), ENTRYKEY, @4(R6)

BNEQ

4\$

MOVQ

INDEX_DESC, -(SP)

MOVQ

ENTRY, -(SP)

CALLS

#4, CALL_USER

BLBC

STATUS, 5\$

MOVL

#1, R0

RET

1505

1508

1506

1509

1512

1513

1514

04 B6

0000V

07

6E

7E

7E

CF

03

50

0C

04

50

66

10

AC

AC

04

50

01

E8

29

12

7D

7D

FB

E9

D0

00045

00048

0004D

0004F

00053

00057

0005C

0005F

00062

3\$:

3\$:

4\$:

5\$:

; .outine Size: 99 bytes, Routine Base: \$CODE\$ + 040F

call_user

F 1
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1 Page 27
(12)

LB
V0

```

703 1515 1 %SBTTL 'call_user';
704 1516 1 ROUTINE call_user (entry, user_routine, index_desc, rfa) =
705 1517 1
706 1518 1 ---
707 1519 1
708 1520 1 This routine is used as an action routine by GET_INDEX
709 1521 1 and SEARCH to call the user with a standard argument
710 1522 1 list for a given key entry.
711 1523 1
712 1524 1 Inputs:
713 1525 1
714 1526 1 entry = Address of key entry
715 1527 1 user_routine = Address of user action routine
716 1528 1 index = Primary index number
717 1529 1
718 1530 1 Outputs:
719 1531 1
720 1532 1 The user routine is called with the following arguments:
721 1533 1 1) If ascii keys, address of key descriptor
722 1534 1 If binary keys, address of longword key
723 1535 1 2) Address of RFA associated with the key
724 1536 1 ---
725 1537 1
726 1538 2 BEGIN
727 1539 2
728 1540 2 MAP
729 1541 2 index_desc: REF BBLOCK, ! Address of index descriptor
730 1542 2 entry: REF BBLOCK; ! Address of key entry
731 1543 2
732 1544 2 BIND
733 1545 2 context = .lbr$gl_control [lbr$l_ctxptr] : BBLOCK;
734 1546 2
735 1547 2 context [ctx$v_found1] = true; ! Flag match found
736 1548 2 IF .index_desc [idd$v_ascii] ! If ASCII keys,
737 1549 2 THEN
738 1550 3 BEGIN
739 1551 3 LOCAL desc: BBLOCK [dsc$s_s_bln]; ! String descriptor
740 1552 3 desc [dsc$w_length] = .entry [idx$b_keylen];
741 1553 3 desc [dsc$a_pointer] = entry [idx$t_keyname];
742 1554 3 perform ((.user_routine) (desc, entry [idx$l_vbn])); ! Call user back
743 1555 3 END
744 1556 2 ELSE
745 1557 2 perform ((.user_routine) (entry [idx$l_keyid], entry [idx$l_vbr]));
746 1558 2
747 1559 2 RETURN true;
748 1560 2
749 1561 1 END;

```

0000 00000 CALL_USER:

SE		08	C2	00002	.WORD	Save nothing
50	0000G	CF	D0	00005	SUBL2	#8, SP
50	OE	A0	D0	0000A	MOVL	LBR\$GL_CONTROL, R0
					MOVL	14(R0), R0

1516

1545

LBR_INDEX
V04=000

call_user

G 1
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 28
(12)

	04	A0	40	8F	88	0000E	BISB2	#64, 4(R0)	:	1547
		50	04	AC	D0	00013	MOVL	ENTRY, R0	:	1552
		12	0C	BC	E9	00017	BLBC	@INDEX_DESC, 1\$	:	1548
		6E	06	A0	9B	0001B	MOVZBW	6(R0), -DESC	:	1552
04	AE	04		07	C1	0001F	ADDL3	#7, ENTRY, DESC+4	:	1553
			04	AC	DD	00025	PUSHL	ENTRY	:	1554
			04	AE	9F	00028	PUSHAB	DESC	:	
				06	11	0002B	BRB	2\$	:	
			04	AC	DD	0002D	PUSHL	ENTRY	:	1557
			06	A0	9F	00030	PUSHAB	6(R0)	:	
08	BC			02	FB	00033	CALLS	#2, @USER_ROUTINE	:	
	03			50	E9	00037	BLBC	STATUS, 3\$	:	
	50			01	D0	0003A	MOVL	#1, R0	:	1559
				04	0003D	3\$:	RET		:	1561

; Routine Size: 62 bytes, Routine Base: \$CODE\$ + 0472

add_key

```
751 1562 1 %SBTTL 'add_key';
752 1563 1 GLOBAL ROUTINE add_key (index, key_desc, key_rfa, stop_vbn) =
753 1564 1 ---
754 1565 1
755 1566 1 This routine adds a key to a specified index. If the index
756 1567 1 block is full, the block is split and a parent index block
757 1568 1 is created and is made to point to the 2 split index blocks.
758 1569 1
759 1570 1 Inputs:
760 1571 1
761 1572 1 index = Primary index number in which key is to be added.
762 1573 1 key_desc = Descriptor of key (ascii or binary) to be added.
763 1574 1 key_rfa = RFA to be associated with key.
764 1575 1 stop_vbn = (Optional) The VBN of an index block in the
765 1576 1 index tree into which the key should be added.
766 1577 1 If not specified, key added at bottom of tree.
767 1578 1
768 1579 1 Outputs:
769 1580 1
770 1581 1 Routine value = Success/failure status code
771 1582 1
772 1583 1 ---
773 1584 2 BEGIN
774 1585 2
775 1586 2 MAP
776 1587 2 key_desc: REF BBLOCK, ! Access as string descriptor
777 1588 2 key_rfa: REF BBLOCK; ! Access as RFA structure
778 1589 2
779 1590 2 LOCAL
780 1591 2 status,
781 1592 2 index_desc: REF BBLOCK, ! Index descriptor
782 1593 2 entry_size, ! Size of each index entry
783 1594 2 index_block1: REF BBLOCK, ! Address of index block
784 1595 2 vbn1, ! VBN of current index block
785 1596 2 genpos, ! Offset to closest entry
786 1597 2 addpos; ! Offset where to add key
787 1598 2
788 1599 2 BUILTIN
789 1600 2 NULLPARAMETER; ! True if argument unspecified
790 1601 2
791 1602 2 MACRO
792 1603 2 entry (address,b) =
793 1604 2 (address+index$c_entries+b)
794 1605 2 %IF %LENGTH GTR 2 %THEN <%REMAINING> %ELSE <0,0,0> %FIX;
795 1606 2
796 1607 2 index_desc = .lbr$gl_control [lbr$l_hdrptr] + lhd$c_idxdesc
797 1608 2 + (.index-1)*idd$c_length;
798 1609 2
799 1610 2 Use false option to check keyword and remove trailing blanks
800 1611 2
801 1612 2 perform (make_upper_case (.key_desc, .key_desc, false));
802 1613 2
803 1614 2 Check for illegal key length if ASCII keys
804 1615 2
805 1616 2 IF .index_desc [idd$v_ascii]
806 1617 2 THEN
807 1618 4 IF ((.key_desc [dsc$w_length] GTR .index_desc [idd$w_keylen]) ! If name too long
```

```
add_key
: 808 1619 3      OR (.key_desc [dsc$w_length] EQL 0))      ! or zero length name
: 809 1620 2      THEN
: 810 1621 2      RETURN lbr$_invkey;                      ! Then return with an error
: 811 1622 2      :
: 812 1623 2      : If no primary index block exists yet, create the block.
: 813 1624 2      :
: 814 1625 2      IF .index_desc [idd$l_vbn] EQL 0          ! If no primary index block yet,
: 815 1626 2      THEN
: 816 1627 3      BEGIN
: 817 1628 3      perform(create_index(vbn1, index_block1)); ! Create index block
: 818 1629 3      index_desc [idd$l_vbn] = .vbn1;          ! Set as root of tree
: 819 1630 3      index_block1 [index$l_parent] = 0;       ! Set backward link
: 820 1631 3      END;
: 821 1632 2      :
: 822 1633 2      : Find the key in the index tree.
: 823 1634 2      :
: 824 1635 2      status = find_key(.index, .key_desc,
: 825 1636 2      (IF NOT NULLPARAMETER(4) THEN .stop_vbn ELSE 0),
: 826 1637 2      vbn1, index_block1, genpos, addpos);
: 827 1638 2      :
: 828 1639 2      : If key found, return duplicate key
: 829 1640 2      :
: 830 1641 2      IF .status                                ! If found,
: 831 1642 2      THEN
: 832 1643 2      RETURN lbr$_dupkey;                      ! Return duplicate key
: 833 1644 2      :
: 834 1645 2      : If the current block is full, split the index block into
: 835 1646 2      : 2 blocks and create a parent index block if necessary.
: 836 1647 2      :
: 837 1648 2      IF .index_desc [idd$v_varlenidx]
: 838 1649 2      THEN
: 839 1650 2      entry_size = idx$c_rfaplsbyt + .key_desc[dsc$w_length]
: 840 1651 2      ELSE
: 841 1652 2      entry_size = idx$c_length + .index_desc [idd$w_keylen];
: 842 1653 2      :
: 843 1654 2      IF .index_block1 [index$w_used] + .entry_size GTRU index$c_blksize
: 844 1655 2      THEN
: 845 1656 3      BEGIN
: 846 1657 3      LOCAL
: 847 1658 3      cur_entry : REF BBLOCK,                  ! step through index entry at a time
: 848 1659 3      last_entry,
: 849 1660 3      last_used,                                ! location of last used byte in index block
: 850 1661 3      move_length,                              ! Length of half the block
: 851 1662 3      ptr,
: 852 1663 3      index_block2: REF BBLOCK,                ! Address of second block
: 853 1664 3      vbn2,                                       ! VBN of second block
: 854 1665 3      rfa2: BBLOCK[rfa$c_length];              ! RFA used by add_key
: 855 1666 3      :
: 856 1667 3      : Create second index and copy about a quarter of the entries into it.
: 857 1668 3      :
: 858 1669 3      :
: 859 1670 3      :
: 860 1671 3      perform(create_index(vbn2, index_block2)); ! Allocate index block
: 861 1672 3      :
: 862 1673 3      IF .index_desc [idd$v_varlenidx]
: 863 1674 3      THEN
: 864 1675 4      BEGIN
```



```
add_key
865 1676 4 cur_entry = .index_block1 + index$entries;
866 1677 4 last_used = .index_block1 + index$entries + .index_block1 [index$w_used];
867 1678 4 DO
868 1679 5 BEGIN
869 1680 5 LOCAL
870 1681 5   entry_len; ! length of variable index entry in index block
871 1682 5   last_entry = .cur_entry;
872 1683 5   entry_len = idx$rfaplsbyt + .cur_entry[idx$b_keylen];
873 1684 5   cur_entry = .cur_entry + .entry_len;
874 1685 5 END
875 1686 5 UNTIL (.cur_entry + lbr$maxkeylen)
876 1687 4   GTR (.index_block1 + index$blksiz);
877 1688 4 move_length = .last_used - .last_entry;
878 1689 4
879 1690 4
880 1691 4   index_block1 [index$w_used] =
881 1692 4     .index_block1 [index$w_used] - .move_length;
882 1693 4   index_block2 [index$w_used] = .move_length;
883 1694 4   CHSMOVE(.move_length, ! Copy half the block
884 1695 4     entry(.index_block1+.index_block1 [index$w_used],0),
885 1696 4     entry(.index_block2,0));
886 1697 4
887 1698 4   reset_highest2(.index,.index_desc,.vbn1,.index_block1); ! Reset highest key
888 1699 4 END
889 1700 3 ELSE ! fixed length keyword storage
890 1701 4 BEGIN
891 1702 4   Move the last fourth of the entries
892 1703 4
893 1704 4   move_length = (.index_block1 [index$w_used] / .entry_size / 4) ! ***
894 1705 5     * .entry_size;
895 1706 4
896 1707 4   If the keyword size is so large that fewer than four keywords fit
897 1708 4   in an index block, then only move out 1 entry.
898 1709 4
899 1710 4   IF .move_length EQL 0 THEN move_length = .entry_size;
900 1711 4   index_block1 [index$w_used] =
901 1712 4     .index_block1 [index$w_used] - .move_length;
902 1713 4   index_block2 [index$w_used] = .move_length;
903 1714 4   CHSMOVE(.move_length, ! Copy 3/4 of the block
904 1715 4     entry(.index_block1+.index_block1 [index$w_used],0),
905 1716 4     entry(.index_block2,0));
906 1717 4
907 1718 4   reset_highest(.index_desc,.vbn1,.index_block1); ! Reset highest key
908 1719 4 END;
909 1720 3
910 1721 3 IF .index_block1 [index$l_parent] EQL 0 ! If at top of tree already,
911 1722 3 THEN
912 1723 3 BEGIN
913 1724 4   Create a parent block for the 2 index blocks.
914 1725 4
915 1726 4   LOCAL
916 1727 4     index_block0: REF BBLOCK, ! Address of parent block
917 1728 4     vbn0; ! VBN of parent block
918 1729 4
919 1730 4   perform(create_index(vbn0, index_block0)); ! Create parent
920 1731 4
921 1732 4
```

```
922 1733 4
923 1734 4 index_block0 [index$l_parent] = .index_block1 [index$l_parent];
924 1735 4 index_block1 [index$l_parent] = .vbn0;
925 1736 4 IF .index_block0 [index$l_parent] EQL 0 ! If root of tree
926 1737 4 THEN
927 1738 4     index_desc [idd$l_vbn] = .vbn0;      ! Reset root pointer
928 1739 4
929 1740 4 IF .index_desc [idd$v_varlenidx]
930 1741 4 THEN
931 1742 5     perform( add_index2(.index, .vbn1, .index_block1) )
932 1743 4 ELSE
933 1744 4     perform( add_index(.index, .vbn1, .index_block1) );
934 1745 4     ! Add highest key to parent
935 1746 3 END;
936 1747 3
937 1748 3 index_block2 [index$l_parent] = .index_block1 [index$l_parent];
938 1749 3
939 1750 3 IF .index_desc [idd$v_varlenidx]
940 1751 3 THEN
941 1752 4 BEGIN
942 1753 4     perform( add_index2(.index, .vbn2, .index_block2) );! Add key to parent
943 1754 4
944 1755 4     ! If any of the entries which were moved into the second
945 1756 4     ! block pointed to sub-indices, reset the parent backpointer
946 1757 4     ! in that sub-index to point to the second block (vbn2).
947 1758 4
948 1759 4     ptr = .index_block2 + index$c_entries;
949 1760 4     last_used = .index_block2 + index$c_entries + .index_block2[index$w_used];
950 1761 4     WHILE .ptr LSS .last_used DO
951 1762 5         BEGIN
952 1763 5             MAP
953 1764 5             ptr: REF BBLOCK;      ! Address index entry
954 1765 5             IF .ptr [idx$w_offset] EQL rfa$c_index ! If points to index,
955 1766 5             THEN
956 1767 6                 BEGIN
957 1768 6                     LOCAL block: REF BBLOCK;
958 1769 6                     perform(find_index(.ptr [idx$l_vbn], block));
959 1770 6                     block [index$l_parent] = .vbn2; ! Reset parent block
960 1771 6                     mark_dirty(.ptr [idx$l_vbn]); ! Mark block dirty
961 1772 5                 END;
962 1773 5                 ptr = .ptr + idx$c_rfaplsbyt + .ptr[idx$b_keylen];
963 1774 4             END;
964 1775 4         END
965 1776 3     ELSE
966 1777 4         BEGIN
967 1778 4             perform( add_index(.index, .vbn2, .index_block2) );! Add key to parent
968 1779 4
969 1780 4             ! If any of the entries which were moved into the second
970 1781 4             ! block pointed to sub-indices, reset the parent backpointer
971 1782 4             ! in that sub-index to point to the second block (vbn2).
972 1783 4
973 1784 4             INCRU ptr FROM .index_block2+index$c_entries
974 1785 4             TO .index_block2+index$c_entries+.index_block2 [index$w_used]-1
975 1786 4             BY .entry_size
976 1787 4         DO
977 1788 5             BEGIN
978 1789 5                 MAP
```

```
add_key
: 979 1790 5 ptr: REF BBLOCK; ! Address index entry
: 980 1791 5 IF .ptr [idx$w_offset] EQL rfa$c_index ! If points to index,
: 981 1792 5 THEN
: 982 1793 6 BEGIN
: 983 1794 6 LOCAL block: REF BBLOCK;
: 984 1795 6 perform(find_index(.ptr [idx$l_vbn], block));
: 985 1796 6 block [index$l_parent] = .vbn2; ! Reset parent block
: 986 1797 6 mark_dirty(.ptr [idx$l_vbn]); ! Mark block dirty
: 987 1798 5 END;
: 988 1799 4 END;
: 989 1800 3 END;
: 990 1801 3
: 991 1802 3 If the add position was in the second half of the
: 992 1803 3 split block, then reset index_block1 and vbn1 so
: 993 1804 3 that the following code adds the key to the second
: 994 1805 3 block. In addition, if we are adding a subindex key,
: 995 1806 3 then adjust the parent block of that subindex to point
: 996 1807 3 to this newly split second block rather than the original
: 997 1808 3 first block. Adjust the add offset for the second block.
: 998 1809 3
: 999 1810 3 IF .addpos GTRU .index_block1 [index$w_used] ! If in 2nd half,
1000 1811 3 THEN
1001 1812 4 BEGIN
1002 1813 4 IF .key_rfa [rfa$w_offset] EQL rfa$c_index ! If index pointer,
1003 1814 4 THEN
1004 1815 5 BEGIN
1005 1816 5 LOCAL block: REF BBLOCK;
1006 1817 5 perform(find_index(.key_rfa [rfa$l_vbn], block));
1007 1818 5 block [index$l_parent] = .vbn2; ! Reset parent block
1008 1819 5 mark_dirty(.key_rfa [rfa$l_vbn]); ! Mark block modified
1009 1820 4 END;
1010 1821 4 mark_dirty(.vbn1); ! Mark block 1 modified now
1011 1822 4 ! since 2 will be marked below
1012 1823 4 addpos = .addpos - .index_block1 [index$w_used]; ! Adjust offset
1013 1824 4 index_block1 = .index_block2; ! Add key to second block
1014 1825 4 vbn1 = .vbn2;
1015 1826 4 END;
1016 1827 3
1017 1828 3
1018 1829 2 END;
1019 1830 2
1020 1831 2 Make room for new entry by pushing all
1021 1832 2 the following entries in use down one.
1022 1833 2
1023 1834 2 CH$MOVE(.index_block1 [index$w_used] - .addpos,
1024 1835 2 entry(.index_block1+.addpos,0),
1025 1836 2 entry(.index_block1+.addpos+.entry_size,0));
1026 1837 2 index_block1 [index$w_used] = .index_block1 [index$w_used]+.entry_size;
1027 1838 2
1028 1839 2 Add the key to the index
1029 1840 2
1030 1841 2 entry(.index_block1+.addpos,idx$l_vbn) = .key_rfa [rfa$l_vbn];
1031 1842 2 entry(.index_block1+.addpos,idx$w_offset) = .key_rfa [rfa$w_offset];
1032 1843 2
1033 1844 2 IF .index_desc [idd$v_ascii] ! If ASCII keys,
1034 1845 2 THEN
1035 1846 3 BEGIN
```

```
1036 1847 3 |
1037 1848 3 | If keywords in this index are to be upper cased for
1038 1849 3 | entry then upcase.
1039 1850 3 |
1040 1851 3 | IF NOT .index_desc [idd$v_nocasentr]
1041 1852 3 | THEN perform (make_upper_case (.key_desc, .key_desc, true));
1042 1853 3 |
1043 1854 3 | CHSMOVE(.key_desc [dsc$w_length], ! Copy ASCII key
1044 1855 3 | .key_desc [dsc$a_pointer],
1045 1856 3 | entry(.index_block1+.addpos,idx$t_keyname));
1046 1857 3 | entry(.index_block1+.addpos,idx$b_keylen) =
1047 1858 3 | .key_desc [dsc$w_length];
1048 1859 3 | END
1049 1860 2 | ELSE ! If binary keys,
1050 1861 2 | entry(.index_block1+.addpos,idx$l_keyid) =
1051 1862 2 | ..key_desc;
1052 1863 2 |
1053 1864 2 | Mark index block modified to be written back later.
1054 1865 2 |
1055 1866 2 | mark_dirty(.vbn1); ! Mark index block modified
1056 1867 2 |
1057 1868 2 | Reset highest keys in parent index blocks.
1058 1869 2 |
1059 1870 2 | IF .addpos+.entry_size EQL .index_block1 [index$w_used]
1060 1871 2 | THEN
1061 1872 2 | IF .index_desc[idd$v_varlenidx] ! If index block has variable length keys
1062 1873 2 | THEN
1063 1874 3 | perform( reset_highest2 (.index, .index_desc, .vbn1, .index_block1))
1064 1875 2 | ELSE
1065 1876 2 | perform( reset_highest (.index_desc, .vbn1, .index_block1) );
1066 1877 2 |
1067 1878 2 |
1068 1879 2 | Unless the entry points to an index, update the index entry total
1069 1880 2 |
1070 1881 3 | BEGIN
1071 1882 3 | BIND
1072 1883 3 | header = .lbr$gl_control[lbr$l_hdrptr] : BBLOCK;
1073 1884 3 |
1074 1885 3 | IF .key_rfa[rfa$w_offset] NEQ rfa$r_index
1075 1886 4 | THEN BEGIN
1076 1887 4 | header[lhd$l_idxcnt] = .header[lhd$l_idxcnt] + 1;
1077 1888 4 |
1078 1889 4 | IF .index EQL 1 ! If index 1
1079 1890 4 | THEN header[lhd$l_modcnt] = .header[lhd$l_modcnt] + 1;
1080 1891 4 | END
1081 1892 3 | ELSE header [lhd$l_idxovh] = .header [lhd$l_idxovh] + 1; ! Count overhead block
1082 1893 2 | END;
1083 1894 2 |
1084 1895 2 | RETURN true;
1085 1896 1 | END;
```

OFFC 00000

.ENTRY ADD_KEY, Save R2,R3,R4,R5,R6,R7,R8,R9,R10,- ; 1563
R11

5E		3L	C2	00002	SUBL2	#60, SP		
50	0000G	CF	D0	00005	MOVL	LBR\$GL_CONTROL, R0		1607
5B	04	AC	D0	0000A	MOVL	INDEX, R11		1608
58	0A	B04B	7E	0000E	MOVAQ	210(R0)[R11], INDEX_DESC		
58	00BC	C8	9E	00013	MOVAB	188(R8), INDEX_DESC		
5A	08	AC	D0	00018	MOVL	KEY_DESC, R10		1612
		52	D4	0001C	CLRL	R2		
51		5A	D0	0001E	MOVL	R10, R1		
50		5A	D0	00021	MOVL	R10, R0		
		0000G	30	00024	BSBW	MAKE_UPPER_CASE		
25		50	E9	00027	BLBC	STATUS, 3\$		
12		68	E9	0002A	BLBC	(INDEX_DESC), 2\$		1616
02	A8	6A	B1	0002D	CMPL	(R10), 2(INDEX_DESC)		1618
		04	1A	00031	BGTRU	1\$		
		6A	B5	00033	TSTW	(R10)		1619
		08	12	00035	BNEQ	2\$		
	50	00000000G	8F	00037	1\$: MOVL	#LBR\$_INVKEY, R0		1621
		04	A8	0003E	RET			
		1A	12	00042	2\$: TSTL	4(INDEX_DESC)		1625
		10	AE	9F	00044	BNEQ	4\$	
		18	AE	9F	00047	PUSHAB	INDEX_BLOCK1	1628
0000V	CF	02	FB	0004A	PUSHAB	VBNI		
	72	50	E9	0004F	CALLS	#2, CREATE_INDEX		
04	A8	14	AE	00052	3\$: BLBC	STATUS, 11\$		
	50	10	AE	00057	MOVL	VBNI, 4(INDEX_DESC)		1629
		02	A0	0005B	MOVL	INDEX_BLOCK1, R0		1630
		08	AE	9F	0005E	4\$: CLRL	2(R0)	
		10	AE	9F	00061	PUSHAB	ADDPOS	1635
		18	AE	9F	00064	PUSHAB	GENPOS	
		20	AE	9F	00067	PUSHAB	INDEX_BLOCK1	
	04	6C	91	0006A	PUSHAB	VBNI		
		0A	1F	0006D	CMPB	(AP), #4		1636
		10	AC	D5	0006F	BLSSU	5\$	
		05	13	00072	TSTL	16(AP)		
		10	AC	DD	00074	BEQL	5\$	
		02	11	00077	PUSHL	STOP_VBN		
		7E	D4	00079	BRB	6\$		
		5A	DD	0007B	5\$: CLRL	-(SP)		
		5B	DD	0007D	6\$: PUSHL	R10		1635
0000V	CF	07	FB	0007F	PUSHL	R11		
	08	50	E9	00084	CALLS	#7, FIND_KEY		
	50	00000000G	8F	00087	BLBC	STATUS, 7\$		1641
			04	0008E	MOVL	#LBR\$_DUPKEY, R0		1643
08	68	02	E1	0008F	RET			
	6E	6A	3C	00093	7\$: BBC	#2, (INDEX_DESC), 8\$		1648
	6E	07	C0	00096	MOVZWL	(R10), ENTRY_SIZE		1650
		07	11	00099	ADDL2	#7, ENTRY_SIZE		
	6E	02	A8	0009B	BRB	9\$		
	6E	06	C0	0009F	8\$: MOVZWL	2(INDEX_DESC), ENTRY_SIZE		1652
	56	10	AE	000A2	ADDL2	#6, ENTRY_SIZE		
	52	66	3C	000A6	9\$: MOVL	INDEX_BLOCK1, R6		1654
	52	6E	C1	000A9	MOVZWL	(R6), R2		
50	8F	50	D1	000AD	ADDL3	ENTRY_SIZE, R2, R0		
		03	1A	000B4	CMPL	R0, #500		
		01BE	31	000B6	BGTRU	10\$		
		18	AE	000B9	BRW	33\$		
					10\$: PUSHAB	INDEX_BLOCK2		1671

			20	AE	9F	000BC	PUSHAB	VBN2		
	0000V	CF		02	FB	000BF	CALLS	#2, CREATE_INDEX		
		01		50	E8	000C4	BLBS	STATUS, 12\$		
					04	000C7	RET			
		57	18	AE	D0	000C8	12\$:	MOVL	INDEX_BLOCK2, R7	1693
		59	14	AE	D0	000CC	12\$:	MOVL	VBN1, R9	1698
49		68		02	E1	000D0	BBC	#2, (INDEX_DESC), 14\$		
		50	0C	A6	9E	000D4	MOVAB	12(R6), CUR_ENTRY		1676
	04	AE	0C	A246	9E	000D8	MOVAB	12(R2)[R6], -LAST_USED		1677
		52	01F4	C6	9E	000DE	MOVAB	500(R6), R2		1687
		53		50	D0	000E3	13\$:	MOVL	CUR_ENTRY, LAST_ENTRY	1682
		51	06	A0	9A	000E6	MOVZBL	6(CUR_ENTRY), ENTRY_LEN		1683
		51		07	C0	000EA	ADDL2	#7, ENTRY_LEN		
		50		51	C0	000ED	ADDL2	ENTRY_LEN, CUR_ENTRY		1684
		51	0080	C0	9E	000F0	MOVAB	128(R0), R1		1686
		52		51	D1	000F5	CMPL	R1, R2		1687
				E9	15	000F8	BLEQ	13\$		
51	04	AE		53	C3	000FA	SUBL3	LAST_ENTRY, LAST_USED, MOVE_LENGTH		1688
		66		51	A2	000FF	SUBW2	MOVE_LENGTH, (R6)		1692
		67		51	50	00102	MOVW	MOVE_LENGTH, (R7)		1693
		50		66	3C	00105	MOVZWL	(R6), R0		1695
OC	A7	OC	A046	51	78	00108	MOV3	MOVE_LENGTH, 12(R0)[R6], 12(R7)		1696
				56	DD	0010F	PUSHL	R6		1698
		7E		58	7D	00111	MOVQ	INDEX_DESC, -(SP)		
				58	DD	00114	PUSHL	R11		
	0000V	CF		04	FB	00116	CALLS	#4, RESET_HIGHEST2		
				29	11	0011B	BRB	16\$		1673
		52		6E	C6	0011D	14\$:	DIVL2	ENTRY_SIZE, R2	1705
		52		04	C6	00120		DIVL2	#4, R2	
51		52		6E	C5	00123		MULL3	ENTRY_SIZE, R2, MOVE_LENGTH	1706
				03	12	00127		BNEQ	15\$	1711
		51		6E	D0	00129		MOVL	ENTRY_SIZE, MOVE_LENGTH	
		66		51	A2	0012C	15\$:	SUBW2	MOVE_LENGTH, (R6)	1713
		67		51	B0	0012F		MOVW	MOVE_LENGTH, (R7)	1714
		50		66	3C	00132		MOVZWL	(R6), R0	1716
OC	A7	OC	A046	51	28	00135	MOV3	MOVE_LENGTH, 12(R0)[R6], 12(R7)		1717
				56	DD	0013C	PUSHL	R6		1719
		7E		58	7D	0013E	MOVQ	INDEX_DESC, -(SP)		
	0000V	CF		03	FB	00141	CALLS	#3, RESET_HIGHEST		
			02	A6	D5	00146	16\$:	TSTL	2(R6)	1722
				45	12	00149		BNEQ	20\$	
			20	AE	9F	0014B		PUSHAB	INDEX_BLOCK0	1732
			28	AE	9F	0014E		PUSHAB	VBN0	
	0000V	CF		02	FB	00151	CALLS	#2, CREATE_INDEX		
		77		50	E9	00156	BLBC	STATUS, 22\$		
		50	20	AE	D0	00159	MOVL	INDEX_BLOCK0, R0		1734
			02	A6	D0	0015D	MOVL	2(R6), 2(R0)		
		02	24	AE	D0	00162	MOVL	VBN0, 2(R6)		1735
			02	A0	D5	00167	TSTL	2(R0)		1736
				05	12	0016A	BNEQ	17\$		
	04	A8	24	AE	D0	0016C	MOVL	VBN0, 4(INDEX_DESC)		1738
0D		68		02	E1	00171	17\$:	BBC	#2, (INDEX_DESC), 18\$	1740
				56	DD	00175		PUSHL	R6	1742
				59	DD	00177		PUSHL	R9	
				58	DD	00179		PUSHL	R11	
	0000V	CF		03	FB	0017B	CALLS	#3, ADD_INDEX2		
				0B	11	00180	BRB	19\$		

			56	DD	00182	18\$:	PUSHL	R6	:	1744
			59	DD	00184		PUSHL	R9	:	
			5B	DD	00186		PUSHL	R11	:	
0000V	CF		03	FB	00188		CALLS	#3, ADD_INDEX	:	
	67		50	E9	0018D	19\$:	BLBC	STATUS, -25\$	:	
02	A7	02	A6	D0	00190	20\$:	MOVL	2(R6), 2(R7)	:	1748
	53	1C	AE	D0	00195		MOVL	VB2, R3	:	1753
4F	68		02	E1	00199		BBC	#2, (INDEX_DESC), 24\$	:	1750
		0088	8F	BB	0019D		PUSHR	#*M<R3,R7>	:	1753
			5B	DD	001A1		PUSHL	R11	:	
0000V	CF		03	FB	001A3		CALLS	#3, ADD_INDEX2	:	
	6F		50	E9	001A8		BLBC	STATUS, -27\$	:	
	52	0C	A7	9E	001AB		MOVAB	12(R7), PTR	:	1759
	50		67	3C	001AF		MOVZWL	(R7), R0	:	1760
04	AE	0C	A047	9E	001B2		MOVAB	12(R0)[R7], LAST_USED	:	
04	AE		52	D1	001B8	21\$:	CMPL	PTR, LAST_USED	:	1761
			75	18	001BC		BGEQ	30\$	:	
FFFF	8F	04	A2	B1	001BE		CMPW	4(PTR), #65535	:	1765
			1B	12	001C4		BNEQ	23\$	:	
	51	28	AE	9E	001C6		MOVAB	BLOCK, R1	:	1769
	50		62	D0	001CA		MOVL	(PTR), R0	:	
		0000V	30	001CD			BSBW	FIND_INDEX	:	
	7E		50	E9	001D0	22\$:	BLBC	STATUS, 31\$	:	
	50	28	AE	D0	001D3		MOVL	BLOCK, R0	:	1770
02	A0		53	D0	001D7		MOVL	R3, 2(R0)	:	
	50		62	D0	001DB		MOVL	(PTR), R0	:	1771
		0000V	30	001DE			BSBW	MARK_DIRTY	:	
	50	06	A2	9A	001E1	23\$:	MOVZBL	6(PTR), R0	:	1773
	52	07	A042	9E	001E5		MOVAB	7(R0)[PTR], PTR	:	
			CC	11	001EA		BRB	21\$	:	1761
		0088	8F	BB	001EC	24\$:	PUSHR	#*M<R3,R7>	:	1778
			5B	DD	001F0		PUSHL	R11	:	
0000V	CF		03	FB	001F2		CALLS	#3, ADD_INDEX	:	
	57		50	E9	001F7	25\$:	BLBC	STATUS, -31\$	:	
	52	0C	A7	9E	001FA		MOVAB	12(R7), R2	:	1784
	50		67	3C	001FE		MOVZWL	(R7), R0	:	1785
	54	0B	A047	9E	00201		MOVAB	11(R0)[R7], R4	:	
			26	11	00206		BRB	29\$	:	1786
FFFF	8F	04	A2	B1	00208	26\$:	CMPW	4(PTR), #65535	:	1791
			1B	12	0020E		BNEQ	28\$	:	
	51	2C	AE	9E	00210		MOVAB	BLOCK, R1	:	1795
	50		62	D0	00214		MOVL	(PTR), R0	:	
		0000V	30	00217			BSBW	FIND_INDEX	:	
	34		50	E9	0021A	27\$:	BLBC	STATUS, 31\$	:	
	50	2C	AE	D0	0021D		MOVL	BLOCK, R0	:	1796
02	A0		53	D0	00221		MOVL	R3, 2(R0)	:	
	50		62	D0	00225		MOVL	(PTR), R0	:	1797
		0000V	30	00228			BSBW	MARK_DIRTY	:	
	52		6E	C0	0022B	28\$:	ADDL2	ENTRY_SIZE, PTR	:	1784
	54		52	D1	0022E	29\$:	CMPL	PTR, R4	:	
			D5	1B	00231		BLEQU	26\$	:	
08	AE	66	00	ED	00233	30\$:	CMPZV	#0, #16, (R6), ADDPOS	:	1810
			3C	1E	00239		BGEQU	33\$	:	
	52	0C	AC	D0	0023B		MOVL	KEY_RFA, R2	:	1813
FFFF	8F	04	A2	B1	0023F		CMPW	4(R2), #65535	:	
			1B	12	00245		BNEQ	32\$	:	
	51	30	AE	9E	00247		MOVAB	BLOCK, R1	:	1817

50			62	DO	0024B	MOVL	(R2), R0				
			0000V	30	0024E	BSBW	FIND_INDEX				
60			50	E9	00251	31\$:	BLBC	STATUS, 34\$			
50		02	AE	DO	00254		MOVL	BLOCK, R0	1818		
50			53	DO	00258		MOVL	R3, 2(R0)			
50			62	DO	0025C		MOVL	(R2), R0	1819		
			0000V	30	0025F		BSBW	MARK_DIRTY			
50			59	DO	00262	32\$:	MOVL	R9, R0	1822		
			0000V	30	00265		BSBW	MARK_DIRTY			
50			66	3C	00268		MOVZWL	(R6), R0	1824		
08			50	C2	0026B		SUBL2	R0, ADDPOS			
10			57	DO	00267		MOVL	R7, INDEX_BLOCK1	1825		
14			53	DO	00273		MOVL	R3, VBN1	1826		
59			AE	DO	00277	33\$:	MOVL	INDEX_BLOCK1, R9	1834		
50			69	3C	0027B		MOVZWL	(R9), R0			
50			AE	C2	0027E		SUBL2	ADDPOS, R0			
59			AE	C1	00282		ADDL3	ADDPOS, R9, R7	1835		
57			OC	C1	00287		ADDL3	#12, ENTRY_SIZE, R6	1836		
56			50	28	0028B		MOVC3	R0, 12(R7), (R6)[R7]			
6647		0C	6E	A0	00291		ADDW2	ENTRY_SIZE, (R9)	1837		
			56	AC	DO	00294	MOVL	KEY_RFA, R6	1841		
		0C	A7	DO	00298		MOVL	(R6), 12(R7)			
		10	A7	04	A6	BO	0029C	MOVW	(R6), 16(R7)	1842	
			1F	E9	002A1		BLBC	(INDEX_DESC), 36\$	1844		
	0F		68	04	E0	002A4	BBS	#4, (INDEX_DESC), 35\$	1851		
			52	01	DO	002A8	MOVL	#1, R2	1852		
			51	5A	DO	002AB	MOVL	R10, R1			
			50	5A	DO	002AE	MOVL	R10, R0			
			0000G	30	002B1		BSBW	MAKE_UPPER_CASE			
			50	E9	002B4	34\$:	BLBC	STATUS, 43\$			
13	A7	04	BA	28	002B7	35\$:	MOVC3	(R10), 24(R10), 19(R7)	1856		
		12	A7	6A	90	002BD	MOVB	(R10), 18(R7)	1858		
			04	11	002C1		BRB	37\$	1844		
		12	A7	6A	DO	002C3	36\$:	MOVL	(R10), 18(R7)	1862	
			50	AE	DO	002C7	37\$:	MOVL	VBN1, R0	1866	
			0000V	30	002CB		BSBW	MARK_DIRTY			
			6E	C1	002CE		ADDL3	ENTRY_SIZE, ADDPOS, R0	1870		
50		08	AE	00	ED	002D3	CMPZV	#0, #16, (R9), R0			
			69	10	12	002D8	BNEQ	40\$			
			10	68	02	E1	002DA	BBC	#2, (INDEX_DESC), 38\$	1872	
				59	DD	002DE	PUSHL	R9	1874		
				18	AE	DD	002E0	PUSHL	VBN1		
					58	DD	002E3	PUSHL	INDEX_DESC		
					5B	DD	002E5	PUSHL	R11		
		0000V	CF	04	FB	002E7	CALLS	#4, RESET_HIGHEST2			
				OC	11	002EC	BRB	39\$			
				59	DD	002EE	38\$:	PUSHL	R9	1876	
				18	AE	DD	002F0	PUSHL	VBN1		
					58	DD	002F3	PUSHL	INDEX_DESC		
		0000V	CF	03	FB	002F5	CALLS	#3, RESET_HIGHEST			
			24	50	E9	002FA	39\$:	BLBC	STATUS, 43\$		
			50	0000G	CF	DO	002FD	40\$:	MOVL	LBR\$GL_CONTROL, R0	1883
			50	0A	A0	DO	00302	MOVL	10(R0), R0		
		FFFF	8F	04	A6	B1	00306	CMPW	4(R6), #65535	1885	
				0D	13	0030C	BEQL	41\$			
				6A	A0	D6	0030E	INCL	106(R0)	1887	
			01	5B	D1	00311	CMPL	R11, #1	1889		

LBR_INDEX
V04=000

add_key

E 2
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VM\$MASTER:[LBR.SRC]INDEX.B32;1

Page 39
(13)

	08	12	00314		BNEQ	42\$	:	
6E	A0	D6	00316		INCL	110(R0)	:	1890
	03	11	00319		BRB	42\$	:	1885
78	A0	D6	0031B	41\$:	INCL	120(R0)	:	1892
50	01	D0	0031E	42\$:	MOVL	#1, R0	:	1895
	04	00321	43\$:	RET			:	1896

; Routine Size: 802 bytes, Routine Base: \$CODE\$ + 04B0

LB
V0

remove_key

```
: 1087 1897 1 %SBTTL 'remove_key';
: 1088 1898 1 GLOBAL ROUTINE remove_key (index, key_desc, stop_vbn) =
: 1089 1899 1
: 1090 1900 1 ---
: 1091 1901 1
: 1092 1902 1 Delete a key from a specified primary index.
: 1093 1903 1
: 1094 1904 1 Inputs:
: 1095 1905 1
: 1096 1906 1 index = Primary index number
: 1097 1907 1 key_desc = Descriptor of key if ASCII, else binary key.
: 1098 1908 1 stop_vbn (optional) = VBN of index block containing key.
: 1099 1909 1
: 1100 1910 1 Outputs:
: 1101 1911 1
: 1102 1912 1 The key is deleted from the index if it exists.
: 1103 1913 1
: 1104 1914 1 true key was found and deleted.
: 1105 1915 1 lbr$_keynotfnd key was not found
: 1106 1916 1 ---
: 1107 1917 1
: 1108 1918 2 BEGIN
: 1109 1919 2
: 1110 1920 2 MAP
: 1111 1921 2 key_desc: REF BBLOCK;
: 1112 1922 2
: 1113 1923 2 LOCAL
: 1114 1924 2 index_desc: REF BBLOCK,      ! Index descriptor
: 1115 1925 2 vbn,                      ! VBN of index block
: 1116 1926 2 index_block: REF BBLOCK,  ! Address of index block
: 1117 1927 2 entry: REF BBLOCK,        ! Address key entry
: 1118 1928 2 offset,                    ! Offset to key entry
: 1119 1929 2 addpos,                    ! Offset to add position
: 1120 1930 2 index_ptr,                ! True if deleting index pointer entry
: 1121 1931 2 entry_size;                ! Size of each entry
: 1122 1932 2
: 1123 1933 2 BUILTIN
: 1124 1934 2 NULLPARAMETER;          ! True if argument unspecified
: 1125 1935 2
: 1126 1936 2
: 1127 1937 2 Find the entry describing the key.
: 1128 1938 2
: 1129 P 1939 2 perform (find_key (.index, .key_desc,
: 1130 P 1940 2 (IF NOT NULLPARAMETER(3) THEN .stop_vbn ELSE 0),
: 1131 1941 2 vbn, index_block, offset, addpos));
: 1132 1942 2
: 1133 1943 2 Push down all following entries in the block.
: 1134 1944 2
: 1135 1945 2 index_desc = .lbr$gl_control [lbr$l_hdrptr] + lhd$_idxdesc
: 1136 1946 2 + (.index-1)*idd$_length;
: 1137 1947 2
: 1138 1948 2 IF .index_desc[idd$_varlenidx] ! If index block has variable length keys
: 1139 1949 2 THEN
: 1140 1950 2 entry_size = idx$_rfaplsbyt + .key_desc [dsc$_length]
: 1141 1951 2 ELSE
: 1142 1952 2 entry_size = idx$_length + .index_desc [idd$_keylen];
: 1143 1953 2
```

remove_key

```
1144 1954 2 entry = .index_block + index$c_entries + .offset;  
1145 1955 2 index_ptr = (.entry[rfa$w_offset] EQL rfa$c_index);  
1146 1956 2  
1147 1957 2 index_block [index$w_used] = .index_block [index$w_used] - .entry_size;  
1148 1958 2 CH$MOVE(.index_block [index$w_used] - .offset,  
1149 1959 2 .entry, .entry_size,  
1150 1960 2 .entry);  
1151 1961 2  
1152 1962 2 ! If the block becomes empty, remove it from the tree.  
1153 1963 2  
1154 1964 2 IF .index_block [index$w_used] EQL 0  
1155 1965 2 THEN  
1156 1966 2 BEGIN  
1157 1967 2 IF .index_block [index$l_parent] EQL 0 ! If root of tree,  
1158 1968 2 THEN  
1159 1969 2 index_desc [idd$l_vbn] = 0 ! Reset tree header  
1160 1970 2 ELSE  
1161 1971 2 remove_key(.index, ! Else, remove parent pointer  
1162 1972 2 .key_desc, .index_block [index$l_parent]);  
1163 1973 2 delete_index(.vbn); ! Deallocate index block  
1164 1974 2 END  
1165 1975 2 ELSE  
1166 1976 2 BEGIN  
1167 1977 2 mark_dirty(.vbn); ! Mark block modified  
1168 1978 2 IF .index_desc [idd$v_varlenidx] ! If index block has variable length keys  
1169 1979 2 THEN  
1170 1980 2 reset_highest2(.index, .index_desc, .vbn, .index_block)  
1171 1981 2 ELSE  
1172 1982 2 reset_highest(.index_desc, .vbn, .index_block);  
1173 1983 2 END;  
1174 1984 2  
1175 1985 2 ! Unless we just removed an index pointer, update index totals in header  
1176 1986 2  
1177 1987 2 BEGIN  
1178 1988 2 BIND  
1179 1989 2 header = .lbr$gl_control[lbr$l_hdrptr] : BBLOCK;  
1180 1990 2  
1181 1991 2 IF NOT .index_ptr  
1182 1992 2 THEN BEGIN  
1183 1993 2 IF .index EQL 1  
1184 1994 2 THEN header[lhd$l_modcnt] = .header[lhd$l_modcnt] - 1;  
1185 1995 2 header[lhd$l_idxcnt] = .header[lhd$l_idxcnt] - 1;  
1186 1996 2 END  
1187 1997 2 ELSE header[lhd$l_idxovh] = .header[lhd$l_idxovh] - 1;  
1188 1998 2 END;  
1189 1999 2  
1190 2000 2 RETURN true;  
1191 2001 2  
1192 2002 1 END;
```

```
OFFC 00000 .ENTRY REMOVE KEY, Save R2,R3,R4,R5,R6,R7,R8,R9,- ; 1898  
5E 10 C2 00002 SUBL2 R10,R11 #16, SP ;
```

LBR_INDEX
V04=000

remove_key

H 2

16-Sep-1984 01:56:12

14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742

DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 42

(14)

LB
V0

			08	5E	DD	00005	PUSHL	SP		1941
			10	AE	9F	00007	PUSHAB	OFFSET		
			18	AE	9F	0000A	PUSHAB	INDEX_BLOCK		
		03		AE	9F	0000D	PUSHAB	VBN		
				6C	91	00010	CMPB	(AP), #3		
				0A	1F	00013	BLSSU	1\$		
			0C	AC	D5	00015	TSTL	12(AP)		
				05	13	00018	BEQL	1\$		
			0C	AC	DD	0001A	PUSHL	STOP_VBN		
				02	11	0001D	BRB	2\$		
				7E	D4	0001F	CLRL	-(SP)		
		58	08	AC	DD	00021	PUSHL	KEY_DESC		
			04	AC	DD	00024	MOVL	INDEX, R8		
				58	DD	00028	PUSHL	R8		
	0000V	CF		07	FB	0002A	CALLS	#7, FIND_KEY		
		01		50	E8	0002F	BLBS	STATUS, 3\$		
					04	00032	RET			
		50	0000G	CF	DD	00033	MOVL	LBR\$GL CONTROL, R0		1945
		57		0A	B048	7E	MOVAQ	@10(R0)[R8], INDEX_DESC		1946
		57	008C	C7	9E	0003D	MOVAB	188(R7), INDEX_DESC		
09		67		02	E1	00042	BBC	#2, (INDEX_DESC), 4\$		1948
		52	08	BC	3C	00046	MOVZWL	@KEY_DESC, ENTRY_SIZE		1950
		52		07	C0	0004A	ADDL2	#7, ENTRY_SIZE		
				07	11	0004D	BRB	5\$		
		52	02	A7	3C	0004F	MOVZWL	2(INDEX_DESC), ENTRY_SIZE		1952
		52		06	C0	00053	ADDL2	#6, ENTRY_SIZE		
		56	08	AE	DD	00056	MOVL	INDEX_BLOCK, R6		1954
50		56	04	AE	C1	0005A	ADDL3	OFFSET, R6, R0		
		50		0C	C0	0005F	ADDL2	#12, ENTRY		
				51	D4	00062	CLRL	R1		1955
	FFFF	8F	04	A0	B1	00064	CMPW	4(ENTRY), #65535		
				02	12	0006A	BNEQ	6\$		
				51	D6	0006C	INCL	R1		
		59		51	DD	0006E	MOVL	R1, INDEX_PTR		
		66		52	A2	00071	SUBW2	ENTRY_SIZE, (R6)		1957
		51		66	3C	00074	MOVZWL	(R6), R1		1958
		51	04	AE	C2	00077	SUBL2	OFFSET, R1		
60		6240		51	28	0007B	MOV3	R1, (ENTRY_SIZE)[ENTRY], (ENTRY)		1960
				66	B5	00080	TSTW	(R6)		1964
				21	12	00082	BNEQ	9\$		
			02	A6	D5	00084	TSTL	2(R6)		1967
				05	12	00087	BNEQ	7\$		
			04	A7	D4	00089	CLRL	4(INDEX_DESC)		1969
				0D	11	0008C	BRB	8\$		
			02	A6	DD	0008E	PUSHL	2(R6)		1972
			08	AC	DD	00091	PUSHL	KEY_DESC		
				58	DD	00094	PUSHL	R8		1971
	FF65	CF		03	FB	00096	CALLS	#3, REMOVE_KEY		
			0C	AE	DD	0009B	PUSHL	VBN		1973
	0000V	CF		01	FB	0009E	CALLS	#1, DELETE_INDEX		
				27	11	000A3	BRB	11\$		1964
		50	0C	AE	DD	000A5	MOVL	VBN, R0		1977
			0000V	30	000A9	BSBW	MARK_DIRTY			
10		67		02	E1	000AC	BBC	#2, (INDEX_DESC), 10\$		1978
				56	DD	000B0	PUSHL	R6		1980
			10	AE	DD	000B2	PUSHL	VBN		
				57	DD	000B5	PUSHL	INDEX_DESC		

LBR_INDEX
V04=000

remove_key

1 2
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 43
(14)

0000V	CF		58	DD	000B7	PUSHL	R8	:	
			04	FB	000B9	CALLS	#4, RESET_HIGHEST2	:	
			0C	11	000BE	BRB	11\$	:	
		10	56	DD	000C0	PUSHL	R6	:	1982
			AE	DD	000C2	PUSHL	VBN	:	
			57	DD	000C5	PUSHL	INDEX_DESC	:	
0000V	CF		03	FB	000C7	CALLS	#3, RESET_HIGHEST	:	
50		0000G	CF	D0	000CC	MOVL	LBR\$GL_CONTROL, R0	:	1989
50		0A	A0	D0	000D1	MOVL	10(R0), R0	:	
0D			59	E8	000D5	BLBS	INDEX_PTR, 13\$	:	1991
01			58	D1	000D8	CMPL	R8, #T	:	1993
			03	12	000DB	BNEQ	12\$	:	
		6E	A0	D7	000DD	DECL	110(R0)	:	1994
		6A	A0	D7	000E0	DECL	106(R0)	:	1995
			03	11	000E3	BRB	14\$	:	1991
		78	A0	D7	000E5	DECL	120(R0)	:	1997
50			01	D0	000E8	MOVL	#1, R0	:	2000
			04	000EB	RET			:	2002

; Routine Size: 236 bytes, Routine Base: \$CODE\$ + 07D2

LB
V0

```
lookup_key
: 1194      2003 1 %SBTTL 'lookup_key';
: 1195      2004 1 GLOBAL ROUTINE lookup_key (index, key_desc, retrfa) =
: 1196      2005 1
: 1197      2006 1 ---
: 1198      2007 1
: 1199      2008 1     Look up a given key and return the RFA associated with
: 1200      2009 1     the key, if found.
: 1201      2010 1
: 1202      2011 1     Inputs:
: 1203      2012 1
: 1204      2013 1     index = Primary index number
: 1205      2014 1     key_desc = Descriptor of key if ASCII, else binary key.
: 1206      2015 1     retradr = Longword to receive key entry address.
: 1207      2016 1     retvbn (optional) = Longword to receive VBN of index block.
: 1208      2017 1
: 1209      2018 1     Outputs:
: 1210      2019 1
: 1211      2020 1     retradr = Address of key entry if found.
: 1212      2021 1
: 1213      2022 1     true          if key found
: 1214      2023 1     lbr$_keynotfnd if key not found
: 1215      2024 1
: 1216      2025 1 ---
: 1217      2026 1
: 1218      2027 2 BEGIN
: 1219      2028 2
: 1220      2029 2 MAP
: 1221      2030 2     retrfa: REF BBLOCK;          ! Address as RFA structure
: 1222      2031 2
: 1223      2032 2 LOCAL
: 1224      2033 2     vbn,                          ! VBN of index block
: 1225      2034 2     index_block: REF BBLOCK,      ! Address of index block
: 1226      2035 2     offset,                      ! Offset to key entry
: 1227      2036 2     addpos,                      ! Offset to add position
: 1228      2037 2     entry: REF BBLOCK;            ! Address of key entry
: 1229      2038 2
: 1230      2039 2 BUILTIN
: 1231      2040 2     NULLPARAMETER;                ! True if argument unspecified
: 1232      2041 2
: 1233      P 2042 2 perform (find_key (.index, .key_desc, 0,
: 1234      2043 2     vbn, index_block, offset, addpos));
: 1235      2044 2
: 1236      2045 2 entry = .index_block + index$_entries + .offset;
: 1237      2046 2
: 1238      2047 2 IF NOT NULLPARAMETER(3)
: 1239      2048 3 THEN BEGIN
: 1240      2049 3     retrfa [rfa$_vbn] = .entry [idx$_vbn];
: 1241      2050 3     retrfa [rfa$_offset] = .entry [idx$_offset];
: 1242      2051 2     END;
: 1243      2052 2
: 1244      2053 2 RETURN true;
: 1245      2054 2
: 1246      2055 1 END;
```


LBR_INDEX
V04=000

lookup_key

K 2
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 45
(15)

LB
V0

		5E		0000	00000	.ENTRY	LOOKUP KEY, Save nothing	:	2004
			10	C2	00002	SUBL2	#16, SP	:	
			5E	DD	00005	PUSHL	SP	:	2043
			08	AE	9F	PUSHAB	OFFSET	:	
			10	AE	9F	PUSHAB	INDEX_BLOCK	:	
			18	AE	9F	PUSHAB	VBN	:	
			7E	D4	00010	CLRL	-(SP)	:	
			04	AC	7D	MOVQ	INDEX, -(SP)	:	
	0000V	7E		07	FB	CALLS	#7, FIND KEY	:	
		CF		50	E9	BLBC	STATUS, 2\$	:	
		22		AE	C1	ADDL3	OFFSET, INDEX_BLOCK, R0	:	2045
50	08	AE	04	0C	C0	ADDL2	#12, ENTRY	:	
		50		6C	91	CMPB	(AP), #3	:	2047
		03		11	1F	BLSSU	1\$	:	
			0C	AC	D5	TSTL	12(AP)	:	
				0C	13	BEQL	1\$	:	
		51	0C	AC	D0	MOVL	RETRFA, R1	:	2049
		61		60	D0	MOVL	(ENTRY), (R1)	:	
	04	A1	04	A0	B0	MOVW	4(ENTRY), 4(R1)	:	2050
		50		01	D0	MOVL	#1, R0	:	2053
				04	00040	RET		:	2055

1\$: 1\$:
2\$: 2\$:

; Routine Size: 65 bytes, Routine Base: \$CODE\$ + 08BE

traverse_keys

```

: 1248 2056 1 %SBTTL 'traverse_keys';
: 1249 2057 1 GLOBAL ROUTINE traverse_keys (index, action_routine, user_routine, rfa) =
: 1250 2058 1
: 1251 2059 1 ---
: 1252 2060 1
: 1253 2061 1     Traverse a specified primary index in key order
: 1254 2062 1     calling a user action routine for each key.
: 1255 2063 1
: 1256 2064 1     Inputs:
: 1257 2065 1
: 1258 2066 1         index = Primary index numebr
: 1259 2067 1         action_routine = Address of internal action routine
: 1260 2068 1         user_routine = Address of user action routine
: 1261 2069 1         rfa = RFA to pass to action routine
: 1262 2070 1
: 1263 2071 1     Outputs:
: 1264 2072 1
: 1265 2073 1         The user routine is called with the following arguments:
: 1266 2074 1             1) Address of key entry
: 1267 2075 1
: 1268 2076 1     ---
: 1269 2077 1
: 1270 2078 2 BEGIN
: 1271 2079 2
: 1272 2080 2 ROUTINE traverse (index_desc, vbn, action_routine, user_routine, txtrfa) =
: 1273 2081 3 BEGIN
: 1274 2082 3
: 1275 2083 3     Scan all entries in the given index block.
: 1276 2084 3
: 1277 2085 3 MAP
: 1278 2086 3     index_desc: REF BBLOCK;      ! Index descriptor
: 1279 2087 3
: 1280 2088 3 LOCAL
: 1281 2089 3     index_block: REF BBLOCK;      ! Index block address
: 1282 2090 3
: 1283 2091 3 perform (find_index (.vbn, index_block));
: 1284 2092 3
: 1285 2093 3 INCRU entry FROM .index_block+index$c_entries
: 1286 2094 3     TO .index_block+index$c_entries+.index_block[index$w_used]-1
: 1287 2095 3     BY idx$c_length + .index_desc [idd$w_keylen]
: 1288 2096 3 DO
: 1289 2097 4 BEGIN
: 1290 2098 4     MAP entry: REF BBLOCK;
: 1291 2099 4     IF .entry [idx$w_offset] EQL rfa$c_index      ! If subindex,
: 1292 2100 4     THEN
: 1293 2101 4         perform (traverse (.index_desc, .entry [idx$l_vbn],
: 1294 2102 5             .action_routine, .user_routine, .txtrfa))
: 1295 2103 4     ELSE
: 1296 2104 4         perform((.action_routine)(.entry, .user_routine, .index_desc, .txtrfa));
: 1297 2105 3     END;
: 1298 2106 3
: 1299 2107 3 RETURN true;
: 1300 2108 2 END;
```



```
OFFC 00000 TRAVERSE:
      5E      04 C2 00002      .WORD      Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      : 2080
      51      6E 9E 00005      SUBL2      #4, SP
      50      08 AC D0 00008      MOVAB     INDEX_BLOCK, R1      : 2091
      51      50 50 0000C      0000V      MOVL     VBN, R0
      50      00 BE 3C 00012      BSBW      FIND_INDEX
      50      6E C0 00016      BLBC      STATUS, 5$
      55      08 A0 9E 00019      MOVZWL    @INDEX_BLOCK, R0      : 2094
      53      04 AC D0 0001D      ADDL2     INDEX_BLOCK, R0
      54      02 A3 3C 00021      MOVAB     11(R0), R5
      54      06 C0 00025      MOVL     INDEX_DESC, R3      : 2095
      52      6E      0C C1 00028      MOVZWL    2(R3), R4
      52      2D 11 0002C      ADDL2     #6, R4
      FFFF     8F      04 A2 B1 0002E 1$:      ADDL3     #12, INDEX_BLOCK, ENTRY      : 2102
      7E      10 AC 7D 00036      BRB      4$
      7E      0C AC DD 0003A      CMPW     4(ENTRY), #65535      : 2099
      BB      AF      05 FB 00041      BNEQ     2$
      7E      10 AC 7D 00036      MOVQ     USER_ROUTINE, -(SP)      : 2102
      7E      0C AC DD 0003A      PUSHL    ACTION_ROUTINE
      7E      62 DD 0003D      PUSHL    (ENTRY)
      7E      53 DD 0003F      PUSHL    R3
      BB      AF      05 FB 00041      CALLS    #5, TRAVERSE
      7E      0E 11 00045      BRB      3$
      7E      14 AC DD 00047 2$:      PUSHL    TXTRFA      : 2104
      7E      53 DD 0004A      PUSHL    R3
      7E      10 AC DD 0004C      PUSHL    USER_ROUTINE
      7E      52 DD 0004F      PUSHL    ENTRY
      7E      0C BC      04 FB 00051      CALLS    #4, @ACTION_ROUTINE
      7E      08      50 E9 00055 3$:      BLBC     STATUS, 5$
      7E      52      54 C0 00058      ADDL2     R4, ENTRY      : 2093
      7E      55      52 D1 0005B 4$:      CMPL     ENTRY, R5
      7E      CE 1B 0005E      BLEQU    1$
      7E      50      01 D0 00060      MOVL     #1, R0
      7E      04 00063 5$:      RET
```

: Routine Size: 100 bytes, Routine Base: \$CODE\$ + 08FF

```
: 1301      2109 2
: 1302      2110 2 ROUTINE traverse2 (index_desc, vbn, action_routine, user_routine, txtrfa) =
: 1303      2111 3 BEGIN
: 1304      2112 3
: 1305      2113 3      Traverse2 handles indices with variable length keywords.
: 1306      2114 3      Scan all entries in the given index block.
: 1307      2115 3
: 1308      2116 3 MAP
: 1309      2117 3      index_desc: REF BBLOCK;      ! Index descriptor
: 1310      2118 3
: 1311      2119 3 LOCAL
: 1312      2120 3      entry,      ! Traverse each entry in index block
: 1313      2121 3      index_block: REF BBLOCK;      ! Index block address
: 1314      2122 3
: 1315      2123 3 perform (find_index (.vbn, index_block));
: 1316      2124 3
: 1317      2125 3 entry = .index_block+index$entries;
: 1318      2126 3 WHILE .entry LESS .index_block+index$entries+.index_block[index$w_used]-1 DO
```

```
traverse_keys
: 1319      2127 4 BEGIN
: 1320      2128 4 MAP entry: REF BBLOCK;
: 1321      2129 4 IF .entry [idx$w_offset] EQL rfa$sc_index ! If subindex,
: 1322      2130 4 THEN
: 1323      P 2131 4     perform (traverse2 (.index_desc, .entry [idx$l_vbn],
: 1324      2132 5         .action_routine, .user_routine, .txtrfa))
: 1325      2133 4 ELSE
: 1326      2134 4     perform((.action_routine)(.entry, .user_routine, .index_desc, .txtrfa));
: 1327      2135 4     entry = .entry + idx$sc_rfaplsbyt + .entry [idx$b_keylen];
: 1328      2136 4 END;
: 1329      2137 3
: 1330      2138 3 RETURN true;
: 1331      2139 2 END;
```

```
OFFC 00000 TRAVERSE2:
      5E      04 C2 00002      .WORD      Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      : 2110
      51      6E 9E 00005      SUBL2      #4, SP
      50      08 AC D0 00008      MOVAB     INDEX_BLOCK, R1
      0000V 30 0000C      MOVL      VBN, R0
      4D      50 E9 0000F      BSBW      FIND_INDEX
      6E      0C C1 00012      BLBC      STATUS, 5$
      50      0C BE 3C 00016 1$:      ADDL3     #12, INDEX_BLOCK, ENTRY
      50      6E C0 0001A      MOVZWL     @INDEX_BLOCK, R0
      50      0B C0 0001D      ADDL2     INDEX_BLOCK, R0
      50      52 D1 00020      ADDL2     #11, R0
      37 18 00023      CMPL      ENTRY, R0
      FFFF 8F 04 A2 B1 00025      BGEQ      4$
      7E      10 AC 7D 0002D      CMPW      4(ENTRY), #65535
      0C      AC DD 00031      BNEQ      2$
      62 DD 00034      MOVQ      USER_ROUTINE, -(SP)
      04 AC DD 00036      PUSHL     ACTION_ROUTINE
      C3 AF      05 FB 00039      PUSHL     (ENTRY)
      0F 11 0003D      PUSHL     INDEX_DESC
      14 AC DD 0003F 2$:      CALLS     #5, TRAVERSE2
      04 AC DD 00042      BRB      3$
      10 AC DD 00045      PUSHL     TXTRFA
      52 DD 00048      PUSHL     INDEX_DESC
      0C BC      04 FB 0004A      PUSHL     USER_ROUTINE
      0E      50 E9 0004E 3$:      PUSHL     ENTRY
      50      06 A2 9A 00051      CALLS     #4, @ACTION_ROUTINE
      52      07 A0 42 00055      BLBC      STATUS, 5$
      BA 11 0005A      MOVZBL     6(ENTRY), R0
      50      01 D0 0005C 4$:      MOVAB     7(R0)[ENTRY], ENTRY
      04 0005F 5$:      BRB      1$
      RET      #1, R0
      : 2126
      : 2138
      : 2139
```

; Routine Size: 96 bytes, Routine Base: \$CODE\$ + 0963

```
: 1332      2140 2 !
: 1333      2141 2 ! Main body of traverse_keys procedure
: 1334      2142 2 !
```

traverse_keys

```
: 1335      2143 2 LOCAL
: 1336      2144 2     index_desc: REF BBLOCK;           ! Index descriptor
: 1337      2145 2
: 1338      2146 2     index_desc = .lbr$gl_control [lbr$l_hdrptr] + lhd$idxdesc
: 1339      2147 2         + (.index-1)*idd$length;
: 1340      2148 2
: 1341      2149 2     IF .index_desc [idd$l_vbn] EQL 0       ! If empty index,
: 1342      2150 2     THEN
: 1343      2151 2         RETURN true;                     ! return immediately
: 1344      2152 2
: 1345      2153 2     !
: 1346      2154 2     ! Set the lock for the index
: 1347      2155 2
: 1348      2156 2     index_desc [idd$v_locked] = true;
: 1349      2157 2
: 1350      2158 2     IF .index_desc[idd$v_varlenidx] !       If index block has variable length keys
: 1351      2159 2     THEN
: 1352      P 2160 2         perform(traverse2(.index_desc, .index_desc [idd$l_vbn],
: 1353      2161 2             .action_routine,.user_routine, .rfa))
: 1354      2162 2     ELSE
: 1355      P 2163 2         perform(traverse(.index_desc, .index_desc [idd$l_vbn],
: 1356      2164 2             .action_routine,.user_routine, .rfa));
: 1357      2165 2
: 1358      2166 2     ! Clear the lock
: 1359      2167 2
: 1360      2168 2     index_desc [idd$v_locked] = false;
: 1361      2169 2
: 1362      2170 2     RETURN true;
: 1363      2171 2
: 1364      2172 1 END;
```

				0004 0000	.ENTRY	TRAVERSE KEYS, Save R2	: 2057
51	0000G	CF	D0	00002	MOVL	LBR\$GL_CONTROL, R1	: 2146
50	04	AC	D0	00007	MOVL	INDEX, R0	: 2147
52	0A B140	7E	0000B		MOVAQ	@10(R1)[R0], INDEX_DESC	
52	00BC	C2	9E	00010	MOVAB	188(R2), INDEX_DESC	
	04	A2	D5	00015	TSTL	4(INDEX_DESC)	: 2149
		31	13	00018	BEQL	3\$	
62		02	88	0001A	BISB2	#2, (INDEX_DESC)	: 2156
13		02	E1	0001D	BBC	#2, (INDEX_DESC), 1\$	: 2158
62		0C	AC	7D	MOVQ	USER_ROUTINE, -(SP)	: 2161
7E		08	AC	DD	PUSHL	ACTION_ROUTINE	
		04	A2	DD	PUSHL	4(INDEX_DESC)	
			52	DD	PUSHL	INDEX_DESC	
FF6E	CF		05	FB	CALLS	#5, TRAVERSE2	
			11	11	BRB	2\$	
7E		0C	AC	7D	MOVQ	USER_ROUTINE, -(SP)	: 2164
		08	AC	DD	PUSHL	ACTION_ROUTINE	
		04	A2	DD	PUSHL	4(INDEX_DESC)	
			52	DD	PUSHL	INDEX_DESC	
FEF7	CF		05	FB	CALLS	#5, TRAVERSE	
	06		50	E9	BLBC	STATUS, 4\$	
62			02	8A	BICB2	#2, (INDEX_DESC)	: 2168

LBR_INDEX
V04=000

traverse_keys

50

01 D0 0004B 3\$: MOVL #1, R0
04 0004E 4\$: RET

C 3
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 BLISS-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 50
(16)

; 2170
; 2172

; Routine Size: 79 bytes, Routine Base: \$CODE\$ + 09C3

find_key

```

: 1366 2173 1 %SBTTL 'find_key';
: 1367 2174 1 GLOBAL ROUTINE find_key (index, key_desc, stop_vbn,
: 1368 2175 1      retvbn, retblkadr, retgenpos, retaddpos) =
: 1369 2176 1      ---
: 1370 2177 1
: 1371 2178 1      find a given key and return all information concerning
: 1372 2179 1      its position within the index tree. This routine is
: 1373 2180 1      used solely by routines such as add_key, remove_key,
: 1374 2181 1      etc. for the common key search processing.
: 1375 2182 1
: 1376 2183 1      Inputs:
: 1377 2184 1
: 1378 2185 1      index = Primary index number
: 1379 2186 1      key_desc = Descriptor of key if ASCII, else binary key.
: 1380 2187 1      stop_vbn = VBN of specific index block, 0 if bottom of tree.
: 1381 2188 1      retvbn = Longword to receive VBN of index block.
: 1382 2189 1      retblkadr = Longword to receive address of index block.
: 1383 2190 1      retgenpos = Longword to receive offset to generic entry.
: 1384 2191 1      retaddpos = Longword to receive offset to add position.
: 1385 2192 1
: 1386 2193 1      Outputs:
: 1387 2194 1
: 1388 2195 1      retvbn = VBN of index block.
: 1389 2196 1      retblkadr = Address of index block.
: 1390 2197 1      retgenpos = Offset to generically closest key entry.
: 1391 2198 1      retaddpos = Offset to position to add key.
: 1392 2199 1
: 1393 2200 1      true          if key found
: 1394 2201 1      false         if key not found
: 1395 2202 1      ---
: 1396 2203 1
: 1397 2204 2 BEGIN
: 1398 2205 2
: 1399 2206 2 MAP
: 1400 2207 2      key_desc: REF BBLOCK;          ! Access as string descriptor
: 1401 2208 2
: 1402 2209 2 LOCAL
: 1403 2210 2      status,
: 1404 2211 2      keydesc : BBLOCK [dsc$c_s_bln],
: 1405 2212 2      keynambuf : BBLOCK [lbr$c_maxkeylen],
: 1406 2213 2      index_desc: REF BBLOCK,          ! Index descriptor
: 1407 2214 2      index_block: REF BBLOCK,        ! Address of index block
: 1408 2215 2      vbn,                          ! VBN of current index block
: 1409 2216 2      offset,                        ! Offset to closest entry
: 1410 2217 2      addpos;                      ! Offset to add position
: 1411 2218 2
: 1412 2219 2 MACRO
: 1413 2220 2      entry (address,b) =
: 1414 2221 2          (address+index$c_entries+b)
: 1415 2222 2          %IF %LENGTH GTR 2 %THEN <%REMAINING> %ELSE <0,0,0> %FI%;
: 1416 2223 2
: 1417 2224 2      index_desc = .lbr$gl_control [lbr$l_hdrptr] + lhd$c_idxdesc
: 1418 2225 2          + (.index-1)*idd$c_length;
: 1419 2226 2
: 1420 2227 2      !
: 1421 2228 2      Get address of primary index block
: 1422 2229 2      vbn = .index_desc [idd$l_vbn];          ! Top of tree
```

find_key

```
1423 2230 2 |
1424 2231 2 |         If no primary index block exists yet, key not found.
1425 2232 2 |
1426 2233 2 | IF .vbn EQL 0                ! If no primary index block yet,
1427 2234 2 | THEN
1428 2235 2 |     RETURN lbr$_keynotfnd;    ! Return key not found
1429 2236 2 |
1430 2237 2 | keydesc = 0;
1431 2238 2 | keydesc [dsc$_length] = .key_desc [dsc$_length];
1432 2239 2 | keydesc [dsc$_pointer] = keynambuf;
1433 2240 2 |
1434 2241 2 |         If keywords in this index are to be upper cased for comparison then upcase
1435 2242 2 |
1436 2243 2 | IF NOT .index_desc [idd$_nocasecmp]
1437 2244 3 | THEN perform (make_upper_case (.key_desc, keydesc, true))
1438 2245 2 | ELSE
1439 2246 3 | BEGIN
1440 2247 3 |     CH$MOVE (.key_desc [dsc$_length], .key_desc [dsc$_pointer],
1441 2248 3 |         .keydesc [dsc$_pointer]);
1442 2249 3 | END;
1443 2250 2 |
1444 2251 2 |
1445 2252 2 |         If a specific index VBN was specified, start there
1446 2253 2 |
1447 2254 2 | IF .stop_vbn NEQ 0          ! If specified,
1448 2255 2 | THEN
1449 2256 2 |     vbn = .stop_vbn;        ! then use it
1450 2257 2 |
1451 2258 2 |         Search down the subtree until either the bottom is
1452 2259 2 |         reached or an error is detected.
1453 2260 2 |
1454 2261 3 | DO BEGIN
1455 2262 3 |
1456 2263 3 |     Locate the index block to be searched. It will either
1457 2264 3 |     find the block in the index cache or it will be read
1458 2265 3 |     from disk and cached.
1459 2266 3 |
1460 2267 3 |     perform(find_index(.vbn, index_block));
1461 2268 3 |
1462 2269 3 |     Search for position of key within index block.
1463 2270 3 |
1464 2271 3 | IF .index_desc[idd$_varlenidx] !   If index block has variable length keys
1465 2272 3 | THEN
1466 2273 3 |     status = key_search2(.index_desc,.index_block,keydesc,
1467 2274 3 |         offset, addpos)
1468 2275 3 | ELSE
1469 2276 3 |     status = key_search(.index_desc,.index_block,keydesc,
1470 2277 3 |         offset, addpos);
1471 2278 3 |
1472 2279 3 |     If a specific index block was specified, then stop the search.
1473 2280 3 |
1474 2281 3 | IF .stop_vbn EQL .vbn          ! If at specified block,
1475 2282 3 | THEN
1476 2283 3 |     EXITLOOP;                ! then stop search
1477 2284 3 |
1478 2285 3 |     If the entry found by the binary search points to another
1479 2286 3 |     index, then continue searching using that index. If it
```



```
1480 2287 3 1 points to an actual data record, then we have reached the
1481 2288 3 1 bottom of the tree and the search is stopped.
1482 2289 3 1
1483 2290 3 1 IF .offset LSS 0 ! If no closest entry,
1484 2291 3 1 OR .entry(.index_block+.offset,idx$_offset) NEQ rfa$_index
1485 2292 3 1 THEN
1486 2293 3 1 EXITLOOP; ! Then stop search
1487 2294 3 1
1488 2295 3 1 vbn = .entry(.index_block+.offset,idx$_vbn); ! Next index
1489 2296 3 1 END
1490 2297 3 1
1491 2298 3 1 UNTIL false; ! Loop until EXITLOOP
1492 2299 3 1
1493 2300 3 1
1494 2301 3 1 Return index block VBN, address and entry offsets.
1495 2302 3 1
1496 2303 2 1 .retvbn = .vbn;
1497 2304 2 1 .retblkadr = .index_block; ! Return block address
1498 2305 2 1 .retgenpos = .offset; ! Return offset to entry
1499 2306 2 1 .retaddpos = .addpos; ! Return offset to add position
1500 2307 2 1
1501 2308 2 1 IF NOT .status ! If key not found,
1502 2309 2 1 THEN
1503 2310 2 1 RETURN lbr$_keynotfnd; ! Return key not found
1504 2311 2 1
1505 2312 2 1
1506 2313 2 1 Propagate actual length of actual index string back to caller
1507 2314 2 1
1508 2315 2 1 key_desc[dsc$_length] = .keydesc[dsc$_length];
1509 2316 2 1 RETURN true; ! Return successful
1510 2317 2 1
1511 2318 1 1 END;
```

			OFFC 00000		.ENTRY	FIND_KEY, Save R2,R3,R4,R5,R6,R7,R8,R9,R10,-;	2174
					R11		
		5E	FF6C	CE 9E 00002	MOVAB	-148(SP), SP	
		51	0000G	CF D0 00007	MOVL	LBR\$GL_CONTROL, R1	2224
		50	04	AC D0 0000C	MOVL	INDEX, R0	2225
		57	0A B140	7E 00010	MOVAQ	@10(R1)[R0], INDEX_DESC	
		57	00BC	C7 9E 00015	MOVAB	188(R7), INDEX_DESC	
		58	04	A7 D0 0001A	MOVL	4(INDEX_DESC),-VBN	2229
				03 12 0001E	BNEQ	1\$	2233
				00A5 31 00020	BRW	8\$	
			F8	AD D4 00023	CLRL	KEYDESC	2237
			08	AC D0 00026	MOVL	KEY_DESC, R6	2238
		56		66 B0 0002A	MOVW	(R6), KEYDESC	
	F8	AD	0C	AE 9E 0002E	MOVAB	KEYNAMBUF, KEYDESC+4	2239
11	FC	AD		03 E0 00033	BBS	#3, (INDEX_DESC), 2\$	2243
		67		F8 AD 9E 00037	MOVAB	KEYDESC, RT	2244
		51		01 D0 0003B	MOVL	#1, R2	
		52		56 D0 0003E	MOVL	R6, R0	
		50	0000G	30 00041	BSBW	MAKE_UPPER_CASE	
		07		50 E8 00044	BLBS	STATUS, 3\$	

find_key

G 3

16-Sep-1984 01:56:12

14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742

DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 54

(17)

LBI
VO

FC	BD	04	B6		66	28	00047	2\$:	RET		
			52	OC	AC	D0	0004E	3\$:	MOVCL	(R6), @4(R6), @KEYDESC+4	2248
			58		03	13	00052		MOVL	STOP_VBN, R2	2254
			51		52	D0	00054		BEQL	4\$	
			50		6E	9E	00057	4\$:	MOVL	R2, VBN	2256
					58	D0	0005A		MOVAB	INDEX_BLOCK, R1	2267
					0000V	30	0005D		MOVL	VBN, R0	
			74		50	E9	00060		BSBW	FIND_INDEX	
15			67		02	E1	00063		BLBC	STATUS, 10\$	
				04	AE	9F	00067		BBC	#2, (INDEX_DESC), 5\$	2271
				OC	AE	9F	0006A		PUSHAB	ADDPOS	2273
				F8	AD	9F	0006D		PUSHAB	OFFSET	
				OC	AE	DD	00070		PUSHAB	KEYDESC	
					57	DD	00073		PUSHL	INDEX_BLOCK	
		0000V	CF		05	FB	00075		PUSHL	INDEX_DESC	
					13	11	0007A		CALLS	#5, KEY_SEARCH2	
				04	AE	9F	0007C	5\$:	BRB	6\$	
				OC	AE	9F	0007F		PUSHAB	ADDPOS	2276
				F8	AD	9F	00082		PUSHAB	OFFSET	
				OC	AE	DD	00085		PUSHAB	KEYDESC	
					57	DD	00088		PUSHL	INDEX_BLOCK	
		0000V	CF		05	FB	0008A		PUSHL	INDEX_DESC	
			53		50	D0	0008F	6\$:	CALLS	#5, KEY_SEARCH	
			58		52	D1	00092		MOVL	R0, STATUS	
					1C	13	00095		CMPL	R2, VBN	2281
			51	08	AE	D0	00097		BEQL	7\$	
					16	19	0009B		MOVL	OFFSET, R1	2290
50			6E		51	C1	0009D		BLSS	7\$	
	FFFF		8F	10	A0	B1	000A1		ADDL3	R1, INDEX_BLOCK, R0	2291
					0A	12	000A7		CMPW	16(R0), #5535	
50			6E		51	C1	000A9		BNEQ	7\$	
			58	OC	A0	D0	000AD		ADDL3	R1, INDEX_BLOCK, R0	2295
					A4	11	000B1		MOVL	12(R0), VBN	
		10	BC		58	D0	000B3	7\$:	BRB	4\$	2261
		14	BC		6E	D0	000B7		MOVL	VBN, @RETVBN	2303
		18	BC	08	AE	D0	000BB		MOVL	INDEX_BLOCK, @RETVBLKADR	2304
		1C	BC	04	AE	D0	000C0		MOVL	OFFSET, @RETVGENPOS	2305
			08		53	E8	000C5		MOVL	ADDPOS, @RETVADDPOS	2306
			50	00000000G	8F	D0	000C8	8\$:	BLBS	STATUS, 9\$	2308
					04	04	000CF		MOVL	#LBR\$_KEYNOTFND, R0	2310
			66	F8	AD	B0	000D0	9\$:	RET		
			50		01	D0	000D4		MOVW	KEYDESC, (R6)	2315
					04	04	000D7	10\$:	MOVL	#1, R0	2316
									RET		2318

; Routine Size: 216 bytes. Routine Base: \$CODE\$ + CA12

key_search

```
1513 2319 1 %SBTTL 'key_search';
1514 2320 1 ROUTINE key_search (index_desc, index_block, key_desc, genpos, addpos) =
1515 2321 1
1516 2322 1 ---
1517 2323 1
1518 2324 1 This routine searches a specified index block using a binary
1519 2325 1 search and returns the position (offset) within the block
1520 2326 1 where the key should be added (if not found) or its exact
1521 2327 1 position (if found).
1522 2328 1
1523 2329 1 It is also used to run down the index tree to find a given
1524 2330 1 key by searching each index block and using the key found
1525 2331 1 generically using this routine to get to the next index block
1526 2332 1 to be searched (the child).
1527 2333 1
1528 2334 1 Inputs:
1529 2335 1
1530 2336 1 index_desc = Primary index descriptor
1531 2337 1 index_block = Address of the index block
1532 2338 1 key_desc = String descriptor of the key
1533 2339 1 genpos = Longword to receive offset to the entry which is
1534 2340 1 most generically close to the key.
1535 2341 1 addpos (optional) = Longword to receive offset to position
1536 2342 1 where the key should be added in the block.
1537 2343 1
1538 2344 1 Outputs:
1539 2345 1
1540 2346 1 genpos = Offset to generically closest entry.
1541 2347 1 addpos (if specified) = Offset to position to add key.
1542 2348 1
1543 2349 1 Routine value = true if key found, else false.
1544 2350 1 ---
1545 2351 1
1546 2352 2 BEGIN
1547 2353 2
1548 2354 2 MAP
1549 2355 2 index_desc: REF BBLOCK, ! Index descriptor
1550 2356 2 index_block: REF BBLOCK, ! Address of index block
1551 2357 2 key_desc: REF BBLOCK; ! String descriptor
1552 2358 2
1553 2359 2 LOCAL
1554 2360 2 entry_size, ! Size of each index entry
1555 2361 2 test, ! -1 (LSS), 0 (EQL), 1 (GTR)
1556 2362 2 min, ! Lower search limit
1557 2363 2 max, ! Upper search limit
1558 2364 2 i; ! Current entry being searched
1559 2365 2
1560 2366 2 BUILTIN
1561 2367 2 NULLPARAMETER; ! True if argument unspecified
1562 2368 2
1563 2369 2 MACRO
1564 2370 2 entry (i,b,p,s,e) =
1565 2371 2 index_block [index$entries+(i-1)*.entry_size+b,p,s,e]%;
1566 2372 2
1567 2373 2 entry_size = idx$length + .index_desc [idd$w_keylen];
1568 2374 2 min = 1; ! Set min and max limits
1569 2375 2 max = .index_block [index$w_used]/.entry_size;
```

```
key_search

: 1570 2376 2
: 1571 2377 2 IF .max EQL 0 ! If null index block,
: 1572 2378 2 THEN
: 1573 2379 3 BEGIN
: 1574 2380 3 i = 1; ! Add at 1st slot
: 1575 2381 3 test = -1; ! No adjustment, key not found
: 1576 2382 3 END
: 1577 2383 2 ELSE
: 1578 2384 2 DO
: 1579 2385 3 BEGIN
: 1580 2386 3 i = (.min+.max) / 2; ! Calculate middle entry
: 1581 2387 3 IF .index_desc [idd$v_ascii] ! If ASCII keys,
: 1582 2388 3 THEN
: 1583 2389 4 BEGIN
: 1584 2390 4 IF .index_desc [idd$v_upcasntry]
: 1585 2391 4 THEN
: 1586 2392 5 BEGIN
: 1587 2393 5 LOCAL
: 1588 2394 5 entrynambuf : BBLOCK [lbr$c_maxkeylen];
: 1589 2395 5
: 1590 2396 5 moveto_upper_case ( .entry [.i,idx$b_keylen],
: 1591 2397 5 entry [.i,idx$t_keyname], entrynambuf);
: 1592 2398 5 test = CH$COMPARE(.key_desc [dsc$w_length], ! Compare ASCII keys
: 1593 2399 5 .key_desc [dsc$a_pointer],
: 1594 2400 5 entry [.i,idx$b_keylen],
: 1595 2401 5 entrynambuf, 0);
: 1596 2402 5 END
: 1597 2403 4 ELSE
: 1598 2404 4 test = CH$COMPARE(.key_desc [dsc$w_length], ! Compare ASCII keys
: 1599 2405 4 .key_desc [dsc$a_pointer],
: 1600 2406 4 entry [.i,idx$b_keylen],
: 1601 2407 4 entry [.i,idx$t_keyname],0);
: 1602 2408 4 END
: 1603 2409 3 ELSE
: 1604 2410 3 test = .key_desc - .entry [.i, idx$l_keyid];
: 1605 2411 3 IF .test GTR 0
: 1606 2412 3 THEN
: 1607 2413 3 min = .i+1 ! Set to upper half
: 1608 2414 3 ELSE
: 1609 2415 3 max = .i-1; ! Set to lower half
: 1610 2416 3 END
: 1611 2417 3 UNTIL (.test EQL 0) OR (.min GTR .max);
: 1612 2418 2
: 1613 2419 2 IF .test GTR 0 ! If greater than last key
: 1614 2420 2 THEN
: 1615 2421 2 i = .i+1; ! then point after last key
: 1616 2422 2
: 1617 2423 2 IF NOT NULLPARAMETER(5) ! If add position specified,
: 1618 2424 2 THEN
: 1619 2425 2 .addpos = (.i-1) * .entry_size; ! Return offset where to add key
: 1620 2426 2
: 1621 2427 2 !
: 1622 2428 2 If the add position points past the end of the block,
: 1623 2429 2 then adjust the closest entry to point to the last entry
: 1624 2430 2 in the block so that add key has a block to insert the
: 1625 2431 2 key into. Note that if the block is empty, return -1.
: 1626 2432 2
```

```
key_search
: 1627 2433 2 .genpos = (.i-1) * .entry_size;      ! Return offset to closest entry
: 1628 2434 2 IF ..genpos GEQU .index_block [index$w_used] ! If over block,
: 1629 2435 2 THEN
: 1630 2436 2     .genpos = ..genpos - .entry_size;  ! Set to last entry in block
: 1631 2437 2
: 1632 2438 2 RETURN .test EQL 0;                  ! True if key found
: 1633 2439 2
: 1634 2440 1 END;
```

```
OFFC 00000 KEY_SEARCH:
5E FF7C CE 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 : 2320
5B 04 AC D0 00007 MOVAB -132(SP), SP : 2373
7E 02 AB 3C 0000B MOVZWL INDEX_DESC, R11 : 2374
6E 06 C0 0000F ADDL2 #6, ENTRY_SIZE : 2375
54 01 D0 00012 MOVZWL #1, MIN : 2377
59 08 BC 3C 00015 MOVZWL @INDEX_BLOCK, MAX : 2380
59 6E C6 00019 DIVL2 ENTRY_SIZE, MAX : 2381
09 12 0001C BNEQ 1$ : 2386
55 01 D0 0001E MOVZWL #1, I : 2397
5A 01 CE 00021 MNEGL #1, TEST : 2396
0088 31 00024 BRW 9$ : 2387
50 54 59 C1 00027 1$: ADDL3 MAX, MIN, R0 : 2399
55 50 02 C7 0002B DIVL3 #2, R0, I : 2390
04 AE FF A5 9E 0002F MOVAB -1(R5), 4(SP) : 2400
04 AE 6E C5 00034 MULL3 ENTRY_SIZE, 4(SP), R0 : 2398
50 50 08 AC C1 00039 ADDL3 INDEX_BLOCK, R0, R6 : 2387
56 50 08 6B E9 0003E BLBC (R11), 5$ : 2390
57 50 08 AC C1 00041 ADDL3 INDEX_BLOCK, R0, R7 : 2387
53 0C AC D0 00046 MOVZBL KEY_DESC, R3 : 2399
29 6B 05 E1 0004A BBC #5, -(R11), 3$ : 2397
52 08 AE 9E 0004E MOVAB ENRYNAMBUF, R2 : 2390
51 13 A7 9E 00052 MOVAB 19(R7), R1 : 2390
50 12 A6 9A 00056 MOVZBL 18(R6), R0 : 2406
0000G 30 0005A BSBW MOVETO_UPPER_CASE : 2407
50 12 A6 9A 0005D MOVZBL 18(R6), R0 : 2411
56 01 D0 00061 MOVZBL #1, R6 : 2387
50 0C BC 2D 00064 CMPC5 @KEY_DESC, @4(R3), #0, R0, ENRYNAMBUF : 2410
08 08 AE 0006B BGTRU 2$ : 2387
03 1A 0006D SBWC #1, R6 : 2411
56 01 D9 0006F SBWC #1, R6 : 2387
5A 56 D0 00072 2$: MOVZBL R6, TEST : 2410
20 11 00075 BRB 6$ : 2387
50 12 A6 9A 00077 3$: MOVZBL 18(R6), R0 : 2411
58 01 D0 0007B MOVZBL #1, R8 : 2387
50 0C BC 2D 0007E CMPC5 @KEY_DESC, @4(R3), #0, R0, 19(R7) : 2411
13 A7 00085 BGTRU 4$ : 2387
03 1A 00087 SBWC #1, R8 : 2411
58 01 D9 00089 SBWC #1, R8 : 2387
5A 58 D0 0008C 4$: MOVZBL R8, TEST : 2410
06 11 0008F BRB 6$ : 2387
5A 0C BC 12 A6 C3 00091 5$: SUBL3 18(R6), @KEY_DESC, TEST : 2411
06 15 00097 6$: BLEQ 7$ : 2387
```

LBR_INDEX
V04=000

key_search

K 3
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 58
(18)

			54	01	A5	9E	00099		MOVAB	1(R5), MIN	:	2413
					04	11	0009D		BRB	8\$	:	
			59	04	AE	D0	0009F	7\$:	MOVL	4(SP), MAX	:	2415
					5A	D5	000A3	8\$:	TSTL	TEST	:	2418
					08	13	000A5		BEQL	9\$	:	
			59		54	D1	000A7		CMPL	MIN, MAX	:	
					03	14	000AA		BGTR	9\$	:	
					FF	78	31	000AC	BRW	1\$	:	
					5A	D5	000AF	9\$:	TSTL	TEST	:	2420
					02	15	000B1		BLEQ	10\$	:	
					55	D6	000B3		INCL	I	:	2422
			05		6C	91	000B5	10\$:	CMPB	(AP), #5	:	2424
					0E	1F	000B8		BLSSU	11\$	:	
				14	AC	D5	000BA		TSTL	20(AP)	:	
			50		09	13	000BD		BEQL	11\$	:	
		14	BC	FF	A5	9E	000BF		MOVAB	-1(R5), R0	:	2426
			50		6E	C5	000C3		MULL3	ENTRY_SIZE, R0, @ADDPPOS	:	
					55	D7	000C8	11\$:	DECL	R5	:	2433
		10	BC		6E	C5	000CA		MULL3	ENTRY_SIZE, R5, @GENPOS	:	
			55		00	ED	000CF		CMPZV	#0, #16, @INDEX_BLOCK, @GENPOS	:	2434
			10		04	1A	000D6		BGTRU	12\$	:	
					6E	C2	000D8		SUBL2	ENTRY_SIZE, @GENPOS	:	2436
					50	D4	000DC	12\$:	CLRL	R0	:	2438
					5A	D5	000DE		TSTL	TEST	:	
					02	12	000E0		BNEQ	13\$	:	
					50	D6	000E2		INCL	R0	:	
					04	000E4	13\$:	RET			:	2440

; Routine Size: 229 bytes, Routine Base: \$CODE\$ + 0AEA

LB
V0

key_search2

```

: 1636 2441 1 %SBTTL 'key_search2';
: 1637 2442 1 ROUTINE key_search2 (index_desc, index_block, key_desc, genpos, addpos) =
: 1638 2443 1
: 1639 2444 1 ---
: 1640 2445 1
: 1641 2446 1 Key_search2 is a modified key_search to handle indices with
: 1642 2447 1 variable length keywords.
: 1643 2448 1 This routine searches a specified index block using a sequential
: 1644 2449 1 search and returns the position (offset) within the block
: 1645 2450 1 where the key should be added (if not found) or its exact
: 1646 2451 1 position (if found).
: 1647 2452 1
: 1648 2453 1 It is also used to run down the index tree to find a given
: 1649 2454 1 key by searching each index block and using the key found
: 1650 2455 1 generically using this routine to get to the next index block
: 1651 2456 1 to be searched (the child).
: 1652 2457 1
: 1653 2458 1 Inputs:
: 1654 2459 1
: 1655 2460 1 index_desc = Primary index descriptor
: 1656 2461 1 index_block = Address of the index block
: 1657 2462 1 key_desc = String descriptor of the key
: 1658 2463 1 genpos = Longword to receive offset to the entry which is
: 1659 2464 1 most generically close to the key.
: 1660 2465 1 addpos (optional) = Longword to receive offset to position
: 1661 2466 1 where the key should be added in the block.
: 1662 2467 1
: 1663 2468 1 Outputs:
: 1664 2469 1
: 1665 2470 1 genpos = Offset to generically closest entry.
: 1666 2471 1 addpos (if specified) = Offset to position to add key.
: 1667 2472 1
: 1668 2473 1 Routine value = true if key found, else false.
: 1669 2474 1 ---
: 1670 2475 1
: 1671 2476 2 BEGIN
: 1672 2477 2
: 1673 2478 2 MAP
: 1674 2479 2 index_desc: REF BBLOCK, ! Index descriptor
: 1675 2480 2 index_block: REF BBLOCK, ! Address of index block
: 1676 2481 2 key_desc: REF BBLOCK; ! String descriptor
: 1677 2482 2
: 1678 2483 2 LOCAL
: 1679 2484 2 entry_size, ! Size of each index entry
: 1680 2485 2 test, ! -1 (LSS), 0 (EQL), 1 (GTR)
: 1681 2486 2 max, ! offset to end of used index
: 1682 2487 2 last_entry, ! offset to last entry examined
: 1683 2488 2 cur_entry; ! offset to current entry examined
: 1684 2489 2
: 1685 2490 2 BUILTIN
: 1686 2491 2 NULLPARAMETER; ! True if argument unspecified
: 1687 2492 2
: 1688 2493 2 MACRO
: 1689 M 2494 2 entry (i,b,p,s,e) =
: 1690 2495 2 index_block [index$entries+i+b,p,s,e]%;
: 1691 2496 2
: 1692 2497 2 IF NOT .index_desc [idd$v_ascii] ! If not ASCII keys,
```

```
key_search2
: 1693 2498 2 THEN
: 1694 2499 2 RETURN lbr$_intrnlerr; ! key_search2 only for ASCII keys
: 1695 2500 2
: 1696 2501 2 max = .index_block [index$_used];
: 1697 2502 2 test = 1; ! Pre set to key not found
: 1698 2503 2 last_entry = 0; ! pre_set to first entry
: 1699 2504 2 cur_entry = 0; ! pre_set to first entry
: 1700 2505 2
: 1701 2506 2 IF .max EQL 0 ! If null index block,
: 1702 2507 2 THEN
: 1703 2508 3 BEGIN
: 1704 2509 3 test = -1;
: 1705 2510 3 END
: 1706 2511 2 ELSE
: 1707 2512 3 BEGIN
: 1708 2513 3 WHILE (.test GTR 0) AND (.cur_entry LSS .max) DO
: 1709 2514 4 BEGIN
: 1710 2515 4 IF .index_desc [idd$_upcasntry]
: 1711 2516 4 THEN
: 1712 2517 5 BEGIN
: 1713 2518 5 LOCAL
: 1714 2519 5 entrynambuf : BBLOCK [lbr$_maxkeylen];
: 1715 2520 5
: 1716 2521 5 moveto_upper_case (.entry [.cur_entry,idx$_keylen],
: 1717 2522 5 entry [.cur_entry,idx$_keyname], entrynambuf);
: 1718 2523 5
: 1719 2524 5 test = CH$COMPARE(.key_desc [dsc$_length], ! Compare ASCII keys
: 1720 2525 5 .key_desc [dsc$_pointer],
: 1721 2526 5 .entry [.cur_entry,idx$_keylen],
: 1722 2527 5 entrynambuf,0)
: 1723 2528 5 END
: 1724 2529 4 ELSE
: 1725 2530 4 test = CH$COMPARE(.key_desc [dsc$_length], ! Compare ASCII keys
: 1726 2531 4 .key_desc [dsc$_pointer],
: 1727 2532 4 .entry [.cur_entry,idx$_keylen],
: 1728 2533 4 entry [.cur_entry,idx$_keyname],0);
: 1729 2534 5 IF (.test GTR 0)
: 1730 2535 4 THEN
: 1731 2536 5 BEGIN
: 1732 2537 5 last_entry = .cur_entry;
: 1733 2538 5 cur_entry = .cur_entry + idx$_rfaplsbyt + .entry [.cur_entry,idx$_keylen];
: 1734 2539 4 END;
: 1735 2540 3 END; ! While
: 1736 2541 2 END;
: 1737 2542 2
: 1738 2543 2 IF NOT NULLPARAMETER(5) ! If add position specified,
: 1739 2544 2 THEN
: 1740 2545 2 .addpos = .cur_entry; ! Return offset where to add key
: 1741 2546 2
: 1742 2547 2 ! If the add position points past the end of the block,
: 1743 2548 2 then adjust the closest entry to point to the last entry
: 1744 2549 2 in the block so that add key has a block to insert the
: 1745 2550 2 key into. Note that if the block is empty, return -1.
: 1746 2551 2
: 1747 2552 2 .genpos = .cur_entry; ! Return offset to closest entry
: 1748 2553 2 IF .genpos GEQU .index_block [index$_used] ! If over block,
: 1749 2554 2 THEN
```



```
key_search2
: 1750 2555 2 .genpos = .last_entry;      ! Set to last entry in block
: 1751 2556 2
: 1752 2557 2
: 1753 2558 2 Must propagate actual length of actual index entry string
: 1754 2559 2 back to caller
: 1755 2560 2
: 1756 2561 2 IF .test EQL 0
: 1757 2562 2 THEN
: 1758 2563 3 BEGIN
: 1759 2564 3 key_desc[dsc$w_length] = .entry[.cur_entry,idx$b_keylen];
: 1760 2565 3 RETURN true;
: 1761 2566 3 END
: 1762 2567 2 ELSE
: 1763 2568 2 RETURN false;
: 1764 2569 2
: 1765 2570 1 END;
```

```
OFFC 00000 KEY_SEARCH2:
5E      80  AE  9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      2442
08      04  BC  E8 00006 MOVAB -128(SP), SP
50 00000000G 8F  D0 0000A BLBS @INDEX_DESC, 1$      2497
          04  00011 MOVL #LBR$_INTRNLERR, R0      2499
5A      08  BC  3C 00012 1$: MOVZWL @INDEX_BLOCK, MAX      2501
59          01  D0 00016 MOVL #1, TEST      2502
          5B  D4 00019 CLRL LAST_ENTRY      2503
          57  D4 0001B CLRL CUR_ENTRY      2504
          5A  D5 0001D TSTL MAX      2506
          05  12 0001F BNEQ 2$
59          01  CE 00021 MNEGL #1, TEST      2509
          66  11 00024 BRB 8$      2506
56      0C  AC  D0 00026 2$: MOVL KEY_DESC, R6      2525
          59  D5 0002A 3$: TSTL TEST      2513
          5E  15 0002C BLEQ 8$
5A          57  D1 0002E CMPL CUR_ENTRY, MAX
          59  18 00031 BGEQ 8$
54      57  08  AC  C1 00033 ADDL3 INDEX_BLOCK, CUR_ENTRY, R4      2522
53      57  08  AC  C1 00038 ADDL3 INDEX_BLOCK, CUR_ENTRY, R3      2521
26      04  BC  05  E1 0003D BBC #5, @INDEX_DESC, 5$      2515
          52  6E  9E 00042 MOVAB ENRYNAMBUF, R2      2522
          51  13  A4  9E 00045 MOVAB 19(R4), R1
          55  12  A3  9A 00049 MOVZBL 18(R3), R5      2521
          50  55  D0 0004D MOVL R5, R0      2522
          0000G 30 00050 BSBW MOVETO_UPPER_CASE
          54  01  D0 00053 MOVL #1, R4      2524
55      00  04  B6  0C  BC  2D 00056 CMPC5 @KEY_DESC, @4(R6), #0, R5, ENRYNAMBUF
          6E  0005D
          03  1A 0005E BGTRU 4$
          54  01  D9 00060 SBWC #1, R4
          59  54  D0 00063 4$: MOVL R4, TEST
          18  11 00066 BRB 7$
          55  12  A3  9A 00068 5$: MOVZBL 18(R3), R5      2532
          58  01  D0 0006C MOVL #1, R8      2533
```

```
; Routine Size: 192 bytes,    Routine Base: $CODES + 0BCF
```

.....

find_index

```
: 1767 2571 1 %SBTTL 'find_index';
: 1768 2572 1 GLOBAL ROUTINE find_index (vbn, address) : JSB_2 =
: 1769 2573 1
: 1770 2574 1 ---
: 1771 2575 1 This routine locates a specific block in the library
: 1772 2576 1 file and returns the address of the block in memory
: 1773 2577 1 If the block is not currently cached in memory, it
: 1774 2578 1 will be automatically read from disk and added to the
: 1775 2579 1 cache.
: 1776 2580 1
: 1777 2581 1 Inputs:
: 1778 2582 1
: 1779 2583 1 vbn = requested block number in file
: 1780 2584 1 address = Longword to receive address of block
: 1781 2585 1
: 1782 2586 1 Outputs:
: 1783 2587 1
: 1784 2588 1 address = Address of block in memory
: 1785 2589 1 ---
: 1786 2590 1
: 1787 2591 2 BEGIN
: 1788 2592 2
: 1789 2593 2 BIND
: 1790 2594 2 header = .lbr$gl_control[lbr$l_hdrptr] : BBLOCK;
: 1791 2595 2
: 1792 2596 2 LOCAL
: 1793 2597 2 status,
: 1794 2598 2 cache_entry: REF BBLOCK; ! Current cache entry address
: 1795 2599 2
: 1796 2600 2 status = lookup_cache(.vbn, cache_entry); ! Lookup block in cache
: 1797 2601 2
: 1798 2602 2 IF .status ! If found,
: 1799 2603 2 THEN
: 1800 2604 3 BEGIN
: 1801 2605 3 .address = .cache_entry [cache$l_address]; ! Return address
: 1802 2606 3 RETURN true;
: 1803 2607 2 END;
: 1804 2608 2
: 1805 2609 2 !
: 1806 2610 2 ! Attempt to read in multiple blocks if vbn is in the pre-allocated index
: 1807 2611 2
: 1808 2612 2 IF .vbn LEQU .header[lhd$l_hiprusd]
: 1809 2613 3 THEN BEGIN
: 1810 P 2614 3 perform (read_n_block (.vbn, MIN (.lbr$gl_maxidxrd, !Read in some index blocks
: 1811 2615 3 (.header [lhd$l_hiprusd] - .vbn + 1)));
: 1812 2616 3 perform(find_index(.vbn, .address)); !Recurse to lookup in cache
: 1813 2617 3 END
: 1814 2618 3 ELSE BEGIN
: 1815 2619 3 perform(read_block(.vbn,.address)); ! Read from disk
: 1816 2620 3
: 1817 2621 3 perform (add_cache (.vbn, cache_entry));! Add cache list entry
: 1818 2622 3 cache_entry [cache$l_address] = .address;
: 1819 2623 3
: 1820 2624 2 END;
: 1821 2625 2
: 1822 2626 2 RETURN true;
: 1823 2627 2
```

		1C	BB	00000	FIND_INDEX::		
					PUSHR	#^M<R2,R3,R4>	2572
5E		04	C2	00002	SUBL2	#4, SP	
53		50	7D	00005	MOVQ	R0, R3	
50	0000G	CF	D0	00008	MOVL	LBR\$GL_CONTROL, R0	2594
52	0A	A0	D0	0000D	MOVL	10(R0), R2	
51		6E	9E	00011	MOVAB	CACHE_ENTRY, R1	2600
50		53	D0	00014	MOVL	VBN, R0	
	0000G	30	00017	BSBW	LOOKUP_CACHE		
09		50	E9	0001A	BLBC	STATUS, 1\$	2602
50		6E	D0	0001D	MOVL	CACHE_ENTRY, R0	2605
64	08	A0	D0	00020	MOVL	8(R0), (ADDRESS)	
		4B	11	00024	BRB	4\$	2606
62	A2	53	D1	00026	1\$:	CMPL	VBN, 98(R2)
		29	1A	0002A	BGTRU	3\$	2612
52	62	53	C3	0002C	SUBL3	VBN, 98(R2), R2	2615
50	01	A2	9E	00031	MOVAB	1(R2), R0	
51	0000G	CF	D0	00035	MOVL	LBR\$GL_MAXIDXRD, R1	
50		51	D1	0003A	CMPL	R1, R0	
		03	15	0003D	BLEQ	2\$	
51		50	D0	0003F	MOVL	R0, R1	
50		53	D0	00042	2\$:	MOVL	VBN, R0
	0000G	30	00045	BSBW	READ_N_BLOCK		
29		50	E9	00048	BLBC	STATUS, 5\$	
50		53	7D	0004B	MOVQ	VBN, R0	2616
		B0	10	0004E	BSBB	FIND_INDEX	
1E		50	E8	00050	BLBS	STATUS, 4\$	
		1F	11	00053	BRB	5\$	
50		53	7D	00055	3\$:	MOVQ	VBN, R0
	0000G	30	00058	BSBW	READ_BLOCK		2619
16		50	E9	0005B	BLBC	STATUS, 5\$	
51		6E	9E	0005E	MOVAB	CACHE_ENTRY, R1	2621
50		53	D0	00061	MOVL	VBN, R0	
	0000G	30	00064	BSBW	ADD_CACHE		
0A		50	E9	00067	BLBC	STATUS, 5\$	
50	08	6E	D0	0006A	MOVL	CACHE_ENTRY, R0	2622
A0		64	D0	0006D	MOVL	(ADDRESS), 8(R0)	
50		01	D0	00071	4\$:	MOVQ	#1, R0
5E		04	C0	00074	5\$:	ADDL2	#4, SP
		1C	BA	00077	POPR	#^M<R2,R3,R4>	2628
		05	00079	RSB			

; Routine Size: 122 bytes, Routine Base: \$CODE\$ + 0C8F

create_index

```
1826 2629 1 %SBTTL 'create_index';
1827 2630 1 ROUTINE create_index (vbn, address) =
1828 2631 1 ---
1829 2632 1
1830 2633 1 This routine allocates a new index block in the file,
1831 2634 1 initializes it, and returns the rfa and address.
1832 2635 1
1833 2636 1 Inputs:
1834 2637 1
1835 2638 1 None
1836 2639 1
1837 2640 1 Outputs:
1838 2641 1
1839 2642 1 vbn = VBN of newly allocated index block
1840 2643 1 address = Address of index block in memory
1841 2644 1 ---
1842 2645 1
1843 2646 2 BEGIN
1844 2647 2
1845 2648 2 BIND
1846 2649 2 context = .lbr$gl_control[lbr$l_ctxptr] : BBLOCK,
1847 2650 2 header = .lbr$gl_control[lbr$l_hdrptr] : BBLOCK;
1848 2651 2
1849 2652 2 LOCAL
1850 2653 2 cache_entry: REF BBLOCK; ! New cache entry address
1851 2654 2
1852 2655 2
1853 2656 2 Allocate block from index cache if possible
1854 2657 2
1855 2658 2 IF .header[lhd$l_freeidx] NEQ 0
1856 2659 3 THEN BEGIN
1857 2660 3 LOCAL
1858 2661 3 buffer : REF VECTOR[,LONG];
1859 2662 3
1860 2663 3 perform(find_block(.header[lhd$l_freeidx], .address, cache_entry));
1861 2664 3 buffer = ..address;
1862 2665 3 .vbn = .header[lhd$l_freeidx];
1863 2666 3 header[lhd$l_freeidx] = .buffer[0];
1864 2667 3 CH$FILL(0, idx$l_length, .buffer);
1865 2668 3 header[lhd$l_freidxblk] = .header[lhd$l_freidxblk] - 1;
1866 2669 3 IF ..vbn GTRO .header[lhd$l_hiprusd]
1867 2670 3 THEN header[lhd$l_hiprusd] = ..vbn;
1868 2671 3 END
1869 2672 3 ELSE BEGIN
1870 2673 3 perform(alloc_block(.vbn, .address)); ! Allocate a disk block
1871 2674 3
1872 2675 3 Add the allocated block to the index cache
1873 2676 3
1874 2677 3 perform (add_cache (..vbn, cache_entry)); ! Add block to cache list
1875 2678 3 cache_entry [cache$l_address] = ..address;
1876 2679 3
1877 2680 3 Initialize the index block
1878 2681 3
1879 2682 4 BEGIN
1880 2683 4 BIND
1881 2684 4 index_block = ..address: BBLOCK; ! Address index block
1882 2685 4
```

```
create_index
: 1883      2686  4      index_block [index$w_used] = 0;      ! No space used initially
: 1884      2687  3      END;
: 1885      2688  2      END;
: 1886      2689  2      mark_dirty(..vbn);      ! Mark index block modified
: 1887      2690  2
: 1888      2691  2      header[lhd$l_idxblks] = .header[lhd$l_idxblks] + 1;      ! Count another index block
: 1889      2692  2
: 1890      2693  2      context [ctx$v_hdrdirty] = true;      ! Mark header dirty
: 1891      2694  2
: 1892      2695  2      RETURN true;
: 1893      2696  2
: 1894      2697  1      END;
```

```
OFFC 00000 CREATE_INDEX:
5E      04  C2 00002      .WORD      Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11      2630
50      0000G CF D0 00005      SUBL2      #4, SP
58      0E  A0 D0 0000A      MOVL      LBR$GL_CONTROL, R0      2649
56      0A  A0 D0 0000E      MOVL      14(R0), R8
57      04  AC D0 00012      MOVL      10(R0), R6      2650
53      5A  A6 D0 00016      MOVL      VBN, R7      2665
      33  13 0001A      MOVL      90(R6), R3      2658
      6E  9E 0001C      BEQL      1$
52      08  AC D0 0001F      MOVAB     CACHE_ENTRY, R2      2663
51      53  D0 00023      MOVL      ADDRESS, R1
50      0000G 30 00026      MOVL      R3, R0
      50  E9 00029      BSBW      FIND_BLOCK
      BC  D0 0002C      BLBC      STATUS, 3$
      53  D0 00030      MOVL      @ADDRESS, BUFFER      2664
      60  D0 00033      MOVL      R3, (R7)      2665
      00  2C 00037      MOVL      (BUFFER), 90(R6)      2666
      60      0003C      MOVCS     #0, (SP), #0, #6, (BUFFER)      2667
      56  A6 D7 0003D      DECL      86(R6)
      67  D0 00040      MOVL      (R7), R2
      52  D1 00043      CMPL      R2, 98(R6)
      30  1B 00047      BLEQU     2$
      52  D0 00049      MOVL      R2, 98(R6)      2670
      2A  11 0004D      BRB      2$
      08  AC D0 0004F 1$:      MOVL      ADDRESS, R1
      57  D0 00053      MOVL      R7, R0
      0000G 30 00056      BSBW      ALLOC_BLOCK
      50  E9 00059      BLBC      STATUS, 3$
      6E  9E 0005C      MOVAB     CACHE_ENTRY, R1      2677
      67  D0 0005F      MOVL      (R7), R2
      52  D0 00062      MOVL      R2, R0
      0000G 30 00065      BSBW      ADD_CACHE
      50  E9 00068      BLBC      STATUS, 3$
      6E  D0 0006B      MOVL      CACHE_ENTRY, R0      2678
      08  A0 08  BC D0 0006E      MOVL      @ADDRESS, 8(R0)
      50  08  BC D0 00073      MOVL      @ADDRESS, R0      2684
      60  B4 00077      CLRW      (R0)      2686
      52  D0 00079 2$:      MOVL      R2, R0      2689
      0000V 30 0007C      BSBW      MARK_DIRTY
```

LBR_INDEX
V04=000

create_index

G 4
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMMASTER:[LBR.SRC]INDEX.B32;1

Page 67
(21)

LBR
V04

04	A8	66	A6	D6	0007F	INCL	102(R6)	:	2691
	50		08	88	00082	BISB2	#8, 4(R8)	:	2693
			01	D0	00086	MOVL	#1, R0	:	2695
			04	00089	3\$:	RET		:	2697

; Routine Size: 138 bytes, Routine Base: \$CODE\$ + 0D09

: F

delete_index

```
: 1896 2698 1 %SBTTL 'delete_index';
: 1897 2699 1 ROUTINE delete_index (vbn) =
: 1898 2700 1
: 1899 2701 1 ---
: 1900 2702 1
: 1901 2703 1 Deallocate the memory used by an index block and
: 1902 2704 1 remove the cache entry.
: 1903 2705 1
: 1904 2706 1 Inputs:
: 1905 2707 1
: 1906 2708 1 vbn = VBN of index block to delete.
: 1907 2709 1
: 1908 2710 1 Outputs:
: 1909 2711 1
: 1910 2712 1 None
: 1911 2713 1 ---
: 1912 2714 1
: 1913 2715 2 BEGIN
: 1914 2716 2
: 1915 2717 2 BIND
: 1916 2718 2 context = .lbr$gl_control[lbr$l_ctxptr] : BBLOCK,
: 1917 2719 2 header = .lbr$gl_control[lbr$l_hdrptr] : BBLOCK;
: 1918 2720 2
: 1919 2721 2 LOCAL
: 1920 2722 2 blockaddr : REF VECTOR[.LONG],
: 1921 2723 2 status,
: 1922 2724 2 cache_entry : REF BBLOCK;
: 1923 2725 2
: 1924 2726 2 perform(find_block(.vbn, blockaddr, cache_entry)); !Get block in memory
: 1925 2727 2 IF .vbn LEQU .header[lhd$l_hipreal]
: 1926 2728 3 THEN BEGIN
: 1927 2729 3 blockaddr[0] = .header[lhd$l_freeidx];
: 1928 2730 3 header[lhd$l_freeidx] = .vbn;
: 1929 2731 3 header[lhd$l_freidxblk] = .header[lhd$l_freidxblk] + 1;
: 1930 2732 3 cache_entry[cache$v_dirty] = true;
: 1931 2733 3 END
: 1932 2734 2 ELSE perform (dealloc_block (.vbn)); ! Just deallocate block
: 1933 2735 2
: 1934 2736 2 header[lhd$l_idxblks] = .header[lhd$l_idxblks] - 1;
: 1935 2737 2
: 1936 2738 2 context [ctx$v_hdrdirty] = true; ! Mark header dirty
: 1937 2739 2 RETURN true;
: 1938 2740 2
: 1939 2741 1 END;
```

OFFC 00000 DELETE_INDEX:

5E		08	C2	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	: 2699
50	0000G	CF	D0	00005	SUBL2	#8, SP	:
53	0A	A0	7D	0000A	MOVL	LBR\$GL_CONTROL, R0	: 2718
52		6E	9E	0000E	MOVQ	10(R0), R3	: 2719
51	04	AE	9E	00011	MOVAB	CACHE_ENTRY, R2	: 2726
50	04	AC	D0	00015	MOVAB	BLOCKADDR, R1	:
					MOVL	VBN, R0	:

LBR_INDEX
V04=000

delete_index

1 4
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 69
(22)

			0000G	30	00019	BSBW	FIND BLOCK	:	
			50	E9	0001C	BLBC	STATUS, 3\$	:	
5E	31							:	
	A3	04	AC	D1	0001F	CMPL	VDN, 94(R3)	:	2727
			16	1A	00024	BGTRU	1\$	:	
04	BE	5A	A3	D0	00026	MOVL	90(R3), @BLOCKADDR	:	2729
5A	A3	04	AC	D0	0002B	MOVL	VDN, 90(R3)	:	2730
		56	A3	D6	00030	INCL	86(R3)	:	2731
	50		6E	D0	00033	MOVL	CACHE_ENTRY, R0	:	2732
0C	A0		01	88	00036	BISB2	#1, 12(R0)	:	
			0A	11	0003A	BRB	2\$	:	2727
	50	04	AC	D0	0003C	MOVL	VDN, R0	:	2734
			0000G	30	00040	BSBW	DEALLOC_BLOCK	:	
	0A		50	E9	00043	BLBC	STATUS, 3\$	:	
		66	A3	D7	00046	DECL	102(R3)	:	2736
04	A4		08	88	00049	BISB2	#8, 4(R4)	:	2738
	50		01	D0	0004D	MOVL	#1, R0	:	2739
			04	00050	3\$:	RET		:	2741

; Routine Size: 81 bytes, Routine Base: \$CODE\$ + 0D93

add_index

```
1941 2742 1 %SBTTL 'add_index';
1942 2743 1 ROUTINE add_index (index, vbn, index_block) =
1943 2744 1
1944 2745 1 ---
1945 2746 1
1946 2747 1 Create a key which points to the specified index block
1947 2748 1 in the parent index block. The highest key in the
1948 2749 1 current block is used as the key value.
1949 2750 1
1950 2751 1 Inputs:
1951 2752 1
1952 2753 1 vbn = VBN of the index block
1953 2754 1 index = Primary index number
1954 2755 1
1955 2756 1 Outputs:
1956 2757 1
1957 2758 1 None
1958 2759 1 ---
1959 2760 1
1960 2761 2 BEGIN
1961 2762 2
1962 2763 2 MAP
1963 2764 2 index_block: REF BBLOCK; ! Address of index block
1964 2765 2
1965 2766 2 LOCAL
1966 2767 2 entry_size, ! Size of each entry
1967 2768 2 last_entry: REF BBLOCK, ! Last index entry in block
1968 2769 2 index_desc: REF BBLOCK, ! Address of index descriptor
1969 2770 2 rfa: BBLOCK [rfa$length]; ! RFA to be associated with key
1970 2771 2
1971 2772 2
1972 2773 2 index_desc = .lbr$gl_control [lbr$l_hdrptr] + lhd$idxdesc
1973 2774 2 + (.index-1)*idd$length;
1974 2775 2
1975 2776 2 Find the last entry in the index block.
1976 2777 2
1977 2778 2 entry_size = idx$length + .index_desc [idd$w_keylen];
1978 2779 2 last_entry = .index_block + index$entries
1979 2780 2 + .index_block [index$w_used] - .entry_size;
1980 2781 2
1981 2782 2 Setup special RFA which points to this index block.
1982 2783 2
1983 2784 2 rfa [rfa$l_vbn] = .vbn; ! Point to this block
1984 2785 2 rfa [rfa$w_offset] = rfa$index; ! Mark as index pointer
1985 2786 2
1986 2787 2 Add the key to the parent index.
1987 2788 2
1988 2789 2 IF .index_desc [idd$v_ascii] ! If ASCII string keys,
1989 2790 2 THEN
1990 2791 3 BEGIN
1991 2792 3 LOCAL
1992 2793 3 desc: BBLOCK [dsc$length]; ! String descriptor
1993 2794 3
1994 2795 3 desc [dsc$w_length] = .last_entry [idx$b_keylen];
1995 2796 3 desc [dsc$a_pointer] = last_entry [idx$t_keyname];
1996 P 2797 3 perform ( add_key (.index, desc, rfa,
1997 2798 3 .index_block [index$l_parent]) );
```


LBR_INDEX
V04=000

K 4
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 71
(23)

**f

```
: 1998      2799      3      END
: 1999      2800      2      ELSE
: 2000      P 2801      2      perform( add_key (.index,last_entry[idx$1_keyid],rfa,
: 2001      2802      2      .index_block [index$1_parent]) );
: 2002      2803      2
: 2003      2804      2      RETURN true;
: 2004      2805      2
: 2005      2806      1      END;
```

```
0004 00000 ADD_INDEX:
      5E      10      C2 00002      .WORD      Save R2      : 2743
      51      0000G CF D0 00005      SUBL2      #16, SP
      50      04      AC D0 0000A      MOVL      LBR$GL_CONTROL, R1      : 2773
      52      0A B140 7E 0000E      MOVL      INDEX, R0      : 2774
      52      00BC C2 9E 00013      MOVAQ      @10(R1)[R0], INDEX_DESC
      51      02      A2 3C 00018      MOVAB      188(R2), INDEX_DESC
      51      06      C0 0001C      MOVZWL     2(INDEX_DESC), ENTRY_SIZE      : 2778
      50      0C      BC 3C 0001F      ADDL2      #6, ENTRY_SIZE
      50      0C      AC C0 00023      MOVZWL     @INDEX_BLOCK, R0      : 2780
      50      51      C2 00027      ADDL2      INDEX_BLOCK, R0
      50      0C      C0 0002A      SUBL2      ENTRY_SIZE, R0
      08      AE      08      AC D0 0002D      ADDL2      #12, LAST_ENTRY
      0C      AE      01      AE 00032      MOVL      VBN, RFA      : 2784
      51      0C      AC D0 00036      MNEGW      #1, RFA+4      : 2785
      14      62      E9 0003A      MOVL      INDEX_BLOCK, R1      : 2798
      6E      06      A0 9B 0003D      BLBC      (INDEX_DESC), 1$      : 2789
      04      AE      07      A0 9E 00041      MOVZBW     6(LAST_ENTRY), DESC      : 2795
      02      A1      DD 00046      MOVAB      7(R0), DESC+4      : 2796
      0C      AE      9F 00049      PUSHL      2(R1)      : 2798
      08      AE      9F 0004C      PUSHAB     RFA
      09      11      0004F      PUSHAB     DESC
      02      A1      DD 00051 1$:      BRB      2$
      0C      AE      9F 00054      PUSHL      2(R1)      : 2802
      06      A0      9F 00057      PUSHAB     RFA
      04      AC      DD 0005A 2$:      PUSHAB     6(LAST_ENTRY)
      F66A CF      04      FB 0005D      PUSHL      INDEX
      03      50      E9 00062      CALLS      #4, ADD_KEY
      50      01      D0 00065      BLBC      STATUS, -3$
      04      00068 3$:      MOVL      #1, R0      : 2804
      RET      : 2806
```

; Routine Size: 105 bytes, Routine Base: \$CODE\$ + 0DE4

add_index2

```
2007 2807 1 %SBTTL 'add_index2';
2008 2808 1 ROUTINE add_index2 (index, vbn, index_block) =
2009 2809 1
2010 2810 1 ---
2011 2811 1
2012 2812 1       Add index2 is a modified add_index to handle indices
2013 2813 1       with variable length keywords.
2014 2814 1       Create a key which points to the specified index block
2015 2815 1       in the parent index block. The highest key in the
2016 2816 1       current block is used as the key value.
2017 2817 1
2018 2818 1       Inputs:
2019 2819 1
2020 2820 1       vbn = VBN of the index block
2021 2821 1       index = Primary index number
2022 2822 1
2023 2823 1       Outputs:
2024 2824 1
2025 2825 1       None
2026 2826 1 ---
2027 2827 1
2028 2828 2 BEGIN
2029 2829 2
2030 2830 2 MAP
2031 2831 2     index_block: REF BBLOCK;           ! Address of index block
2032 2832 2
2033 2833 2 LOCAL
2034 2834 2     entry_size,           ! Size of each entry
2035 2835 2     last_entry: REF BBLOCK, ! Last index entry in block
2036 2836 2     next_entry: REF BBLOCK, ! search for last index entry in block.
2037 2837 2     index_desc: REF BBLOCK, ! Address of index descriptor
2038 2838 2     rfa: BBLOCK [rfa$length]; ! RFA to be associated with key
2039 2839 2
2040 2840 2
2041 2841 2     index_desc = .lbr$gl_control [lbr$l_hdrptr] + lhd$idxdesc
2042 2842 2               + (.index-1)*idd$length;
2043 2843 2
2044 2844 2     Find the last entry in the index block.
2045 2845 2
2046 2846 2     last_entry = .index_block + index$entries;
2047 2847 2     next_entry = .last_entry;
2048 2848 2     WHILE .next_entry [SS .index_block+index$entries+.index_block[index$w_used] DO
2049 2849 3       BEGIN
2050 2850 3         last_entry = .next_entry;
2051 2851 3         next_entry = .next_entry + idx$rfaplsbyt + .next_entry[idx$b_keylen];
2052 2852 3       END;
2053 2853 2
2054 2854 2     Setup special RFA which points to this index block.
2055 2855 2
2056 2856 2     rfa [rfa$l_vbn] = .vbn;           ! Point to this block
2057 2857 2     rfa [rfa$w_offset] = rfa$index;   ! Mark as index pointer
2058 2858 2
2059 2859 2     Add the key to the parent index.
2060 2860 2
2061 2861 2     IF .index_desc [idd$v_ascii]      ! If ASCII string keys,
2062 2862 2     THEN
2063 2863 3       BEGIN
```

```

: 2064      2864 3      LOCAL
: 2065      2865 3      desc: BBLOCK [dsc$c_s_bln];      ! String descriptor
: 2066      2866 3
: 2067      2867 3      desc [dsc$w_length] = .last_entry [idx$b_keylen];
: 2068      2868 3      desc [dsc$a_pointer] = last_entry [idx$t_keyname];
: 2069      2869 3      perform( add_key (.index, desc, rfa,
: 2070      2870 3      .index_block [index$_parent]) );
: 2071      2871 3      END
: 2072      2872 2 ELSE
: 2073      2873 2      RETURN lbr$_intrnlerr;      ! add_index2 only for ASCII keys
: 2074      2874 2
: 2075      2875 2      RETURN true;
: 2076      2876 2
: 2077      2877 1      END;
```

```

                                001C 00000 ADD_INDEX2:
                                .WORD      Save R2,R3,R4      : 2808
                                SUBL2      #16, SP
                                MOVL      LBR$GL_CONTROL, R1   : 2841
                                MOVL      INDEX, R0            : 2842
                                MOVAQ     @10(R1)(R0), INDEX_DESC
                                MOVAB     188(R4), INDEX_DESC
                                MOVL      INDEX_BLOCK, R3      : 2846
                                MOVAB     12(R3), LAST_ENTRY
                                MOVL      LAST_ENTRY, NEXT_ENTRY : 2847
                                MOVZWL    (R3)-R1             : 2848
                                MOVAB     12(R3)(R1), R1
                                CMPL      NEXT_ENTRY, R1
                                BGEQ      2$
                                MOVL      NEXT_ENTRY, LAST_ENTRY : 2850
                                MOVZBL    6(NEXT_ENTRY), R1    : 2851
                                MOVAB     7(R1)(NEXT_ENTRY), NEXT_ENTRY
                                BRB       1$
                                MOVL      VBN, RFA             : 2848
                                MNEGW     #1, RFA+4           : 2856
                                BLBC      (INDEX_DESC), 3$     : 2857
                                MOVZBW    6(LAST_ENTRY), DESC  : 2861
                                MOVAB     7(R0), -DESC+4       : 2867
                                PUSHL     2(R3)                 : 2868
                                PUSHAB    RFA                  : 2870
                                PUSHAB    DESC
                                PUSHL     INDEX
                                CALLS     #4, ADD_KEY
                                BLBS      STATUS, -4$
                                RET
                                MOVL      #LBR$_INTRNLERR, R0   : 2873
                                RET
                                MOVL      #1, R0                : 2875
                                RET                                : 2877

08  AE      08  AC  D0 0003E 2$:
0C  AE      01  AE 00043
1E      64  E9 00047
04  AE      06  A0 9B 0004A
      07  A0 9E 0004E
      02  A3 DD 00053
      0C  AE 9F 00056
      08  AE 9F 00059
      04  AC DD 0005C
F5FF CF      04  FB 0005F
09      50  E8 00064
      04 00067
      50 00000000G 8F D0 00068 3$:
      04 0006F
      50      01 D0 00070 4$:
      04 00073
```

; Routine Size: 116 bytes, Routine Base: \$CODE\$ + 0E4D

```
reset_highest

: 2079 2878 1 %SBTTL 'reset_highest';
: 2080 2879 1 ROUTINF reset_highest (index_desc, vbn, index_block) =
: 2081 2880 1
: 2082 2881 1 ---
: 2083 2882 1
: 2084 2883 1      Reset the index pointers in the parent blocks
: 2085 2884 1      pointing to the specified index block. Each
: 2086 2885 1      index pointer in a parent block contains the
: 2087 2886 1      highest key in the subindex block in order for
: 2088 2887 1      binary searches to work. This routine is called
: 2089 2888 1      when the index block has changed in order to
: 2090 2889 1      reset the parents highest keys to the proper value.
: 2091 2890 1
: 2092 2891 1      Inputs:
: 2093 2892 1
: 2094 2893 1      index_desc = Address of primary index descriptor
: 2095 2894 1      vbn = VBN of index block
: 2096 2895 1      index_block = Address of index block
: 2097 2896 1
: 2098 2897 1      Outputs:
: 2099 2898 1
: 2100 2899 1      The highest keys in the parents are reset.
: 2101 2900 1
: 2102 2901 1 ---
: 2103 2902 1
: 2104 2903 2 BEGIN
: 2105 2904 2
: 2106 2905 2 MAP
: 2107 2906 2     index_desc: REF BBLOCK,      ! Address of index descriptor
: 2108 2907 2     index_block: REF BBLOCK;  ! Address of index block
: 2109 2908 2
: 2110 2909 2 LOCAL
: 2111 2910 2     entry_size,      ! Size of each entry
: 2112 2911 2     last_entry: REF BBLOCK,  ! Last index entry in block
: 2113 2912 2     parent_block: REF BBLOCK, ! Address of parent block
: 2114 2913 2     parent_entry: REF BBLOCK;  ! Address of parent entry
: 2115 2914 2
: 2116 2915 2
: 2117 2916 2 IF .index_block [index$l_parent] EQL 0 ! If no parent
: 2118 2917 2 THEN
: 2119 2918 2     RETURN true;      ! then return done
: 2120 2919 2
: 2121 2920 2     Find the last entry in the index block.
: 2122 2921 2
: 2123 2922 2     entry_size = idx$c_length + .index_desc [idd$w_keylen];
: 2124 2923 2     last_entry = .index_block + index$c_entries
: 2125 2924 2     + .index_block [index$w_used] - .entry_size;
: 2126 2925 2
: 2127 2926 2     Find the parent index block.
: 2128 2927 2
: 2129 2928 2     perform (find_index (.index_block [index$l_parent], parent_block));
: 2130 2929 2
: 2131 2930 2     Locate the pointer to the subindex block.
: 2132 2931 2
: 2133 2932 2     INCRU entry FROM .parent_block+index$c_entries BY .entry_size
: 2134 2933 2     DO
: 2135 2934 3     BEGIN
```

```
reset_highest
: 2136      2935 3      MAP
: 2137      2936 3      entry: REF BBLOCK;          ! Address index entry
: 2138      2937 3
: 2139      2938 3      IF .entry [idx$l_vbn] EQL .vbn    ! If points to subindex,
: 2140      2939 3      THEN
: 2141      2940 4      BEGIN
: 2142      2941 4      parent_entry = .entry;          ! Set address of parent entry
: 2143      2942 4      EXITLOOP;                      ! then exit the scan
: 2144      2943 3      END;
: 2145      2944 3      END;
: 2146      2945 2      |
: 2147      2946 2      |      Update the key in the parent index.
: 2148      2947 2      |
: 2149      2948 2      IF .index_desc [idd$v_ascii]      ! If ASCII string keys,
: 2150      2949 2      THEN
: 2151      2950 3      BEGIN
: 2152      2951 3      parent_entry [idx$b_keylen] = .last_entry [idx$b_keylen];
: 2153      2952 3      CH$MOVE(.last_entry [idx$b_keylen], ! Copy ASCII-key
: 2154      2953 3      last_entry [idx$t_keyname],
: 2155      2954 3      parent_entry [idx$t_keyname]);
: 2156      2955 3      END
: 2157      2956 2      ELSE
: 2158      2957 2      parent_entry [idx$l_keyid] = .last_entry [idx$l_keyid];
: 2159      2958 2      |
: 2160      2959 2      |      Mark the parent index block modified.
: 2161      2960 2      |
: 2162      2961 2      mark_dirty(.index_block [index$l_parent]);
: 2163      2962 2      |
: 2164      2963 2      |      Reset the highest key in the parents parent.
: 2165      2964 2      |
: 2166      2965 2      reset_highest(.index_desc,.index_block [index$l_parent],.parent_block);
: 2167      2966 2
: 2168      2967 2      RETURN true;
: 2169      2968 2
: 2170      2969 1      END;
```

OFFC 00000 RESET_HIGHEST:

5E	04	C2	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	: 2879
51	0C	AC	D0 00005	SUBL2	#4, SP	
57	02	A1	D0 00009	MOVL	INDEX_BLOCK, R1	: 2916
		60	13 0000D	MOVL	2(R1), R7	
56	04	AC	D0 0000F	BEQL	6\$	
53	02	A6	3C 00013	MOVL	INDEX_DESC, R6	: 2922
53		06	C0 00017	MOVZWL	2(R6), ENTRY_SIZE	
50		61	3C 0001A	ADDL2	#6, ENTRY_SIZE	
50		51	C1 0001D	MOVZWL	(R1), R0	: 2924
52		53	C2 00021	ADDL3	R1, R0, R2	
52		0C	C0 00024	SUBL2	ENTRY_SIZE, R2	
51		6E	9E 00027	ADDL2	#12, LAST_ENTRY	
50		57	D0 0002A	MOVAB	PARENT_BLOCK, R1	: 2928
	FD9E	30	0002D	MOVL	R7, R0	
3F	50	E9	00030	BSBW	FIND_INDEX	
				BLBC	STATUS, 7\$	

LBR_INDEX
V04=000

reset_highest

C 5
16-Sep-1984 01:56:12
14-Sep-1984 12:37:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[LBR.SRC]INDEX.B32;1

Page 76
(25)

LBR

51		08	6E		0C	C1	00033		ADDL3	#12, PARENT_BLOCK, ENTRY	:	2938	
			AC		61	D1	00037	1\$:	CMPL	(ENTRY), VBR	:		
					05	12	0003B		BNEQ	2\$	:		
			50		51	D0	0003D		MOVL	ENTRY, PARENT_ENTRY	:	2941	
					05	11	00040		BRB	3\$	:	2940	
			51		53	C0	00042	2\$:	ADDL2	ENTRY_SIZE, ENTRY	:	2932	
					F0	11	00045		BRB	1\$	:		
			11		66	E9	00047	3\$:	BLBC	(R6), 4\$	:	2948	
		06	A0		06	A2	90	0004A	MOVB	6(LAST_ENTRY), 6(PARENT_ENTRY)	:	2951	
			51		06	A2	9A	0004F	MOVZBL	6(LAST_ENTRY), R1	:	2952	
07	A0	07	A2		51	28	00053		MOVC3	R1, 7(LAST_ENTRY), 7(PARENT_ENTRY)	:	2954	
					05	11	00059		BRB	5\$	:	2948	
		06	A0		06	A2	D0	0005B	4\$:	MOVL	6(LAST_ENTRY), 6(PARENT_ENTRY)	:	2957
			50		57	D0	00060	5\$:	MOVL	R7, R0	:	2961	
					0000V	30	00063		BSBW	MARK DIRTY	:		
					6E	DD	00066		PUSHL	PARENT_BLOCK	:	2965	
			7E		56	7D	00068		MOVQ	R6, -(SP)	:		
		91	AF		03	FB	0006B		CALLS	#3, RESET_HIGHEST	:		
			50		01	D0	0006F	6\$:	MOVL	#1, R0	:	2967	
					04	00072	7\$:		RET		:	2969	

; Routine Size: 115 bytes, Routine Base: \$CODE\$ + 0EC1

reset_highest2

```
2172 2970 1 %SBTTL 'reset_highest2';
2173 2971 1 ROUTINE reset_highest2 (index, index_desc, vbn, index_block) =
2174 2972 1
2175 2973 1 ---
2176 2974 1
2177 2975 1     Reset_highest2 is a modified reset_highest
2178 2976 1     to handle variable length keyword indices.
2179 2977 1     Reset the index pointers in the parent blocks
2180 2978 1     pointing to the specified index block. Each
2181 2979 1     index pointer in a parent block contains the
2182 2980 1     highest key in the subindex block in order for
2183 2981 1     binary searches to work. This routine is called
2184 2982 1     when the index block has changed in order to
2185 2983 1     reset the parents highest keys to the proper value.
2186 2984 1
2187 2985 1 Inputs:
2188 2986 1
2189 2987 1     index_desc = Address of primary index descriptor
2190 2988 1     vbn = VBN of index block
2191 2989 1     index_block = Address of index block
2192 2990 1
2193 2991 1 Outputs:
2194 2992 1
2195 2993 1     The highest keys in the parents are reset.
2196 2994 1
2197 2995 1 ---
2198 2996 1
2199 2997 2 BEGIN
2200 2998 2
2201 2999 2 MAP
2202 3000 2     index_desc: REF BBLOCK,           ! Address of index descriptor
2203 3001 2     index_block: REF BBLOCK;         ! Address of index block
2204 3002 2
2205 3003 2 LOCAL
2206 3004 2     entry,                             ! index block entry
2207 3005 2     entry_size,                         ! Size of each entry
2208 3006 2     last_entry: REF BBLOCK,             ! Last index entry in block
2209 3007 2     next_entry: REF BBLOCK,             ! search for last index entry in block.
2210 3008 2     parent_block: REF BBLOCK,          ! Address of parent block
2211 3009 2     parent_entry: REF BBLOCK;          ! Address of parent entry
2212 3010 2
2213 3011 2
2214 3012 2 IF .index_block [index$l_parent] EQL 0 ! If no parent
2215 3013 2 THEN
2216 3014 2     RETURN true;                      ! then return done
2217 3015 2
2218 3016 2     Find the last entry in the index block.
2219 3017 2
2220 3018 2 next_entry = .index_block + index$c_entries;
2221 3019 2 WHILE .next_entry LESS .index_block+index$c_entries+.index_block[index$w_used] DO
2222 3020 3     BEGIN
2223 3021 3         last_entry = .next_entry;
2224 3022 3         next_entry = .next_entry + idx$c_rfaplsbyt + .next_entry[idx$b_keylen];
2225 3023 3     END;
2226 3024 2
2227 3025 2     Find the parent index block.
2228 3026 2
```

reset_highest2

```
2229 3027 2 perform (find_index (.index_block [index$l_parent], parent_block));
2230 3028 2
2231 3029 2      Locate the pointer to the subindex block.
2232 3030 2
2233 3031 2 entry = .parent_block+index$c_entries;
2234 3032 2 WHILE true DO
2235 3033 3   BEGIN
2236 3034 3     MAP
2237 3035 3       entry: REF BBLOCK;          ! Address index entry
2238 3036 3
2239 3037 3     IF .entry [idx$l_vbn] EQL .vbn    ! If points to subindex.
2240 3038 3     THEN
2241 3039 4       BEGIN
2242 3040 4         parent_entry = .entry;      ! Set address of parent entry
2243 3041 4         EXITLOOP;                  ! then exit the scan
2244 3042 4       END
2245 3043 3     ELSE
2246 3044 3       entry = .entry + idx$c_rfaplsbyt + .entry [idx$b_keylen];
2247 3045 3       IF .entry GTR .parent_block + lbr$c_pagesize    ! Don't loop forever if not found
2248 3046 3       THEN RETURN lbr$_intrnlerr;
2249 3047 2     END;
2250 3048 2
2251 3049 2      Update the key in the parent index.
2252 3050 2
2253 3051 2 IF .index_desc [idd$v_ascii]      ! If ASCII string keys,
2254 3052 2 THEN
2255 3053 3   BEGIN
2256 3054 3     IF .parent_entry [idx$b_keylen] EQL .last_entry [idx$b_keylen]
2257 3055 3     THEN      ! We're in luck, they are the same size
2258 3056 4       BEGIN
2259 3057 4         parent_entry [idx$b_keylen] = .last_entry [idx$b_keylen];
2260 3058 4         CH$MOVE(.last_entry [idx$b_keylen],      ! Copy ASCII key
2261 3059 4           last_entry [idx$t_keyname],
2262 3060 4           parent_entry [idx$t_keyname]);
2263 3061 4       END
2264 3062 3     ELSE      ! Remove old entry, compress, and enter new one.
2265 3063 4       BEGIN
2266 3064 4         LOCAL
2267 3065 4           parent_entry_siz;
2268 3066 4
2269 3067 4         parent_entry_siz = idx$c_rfaplsbyt + .parent_entry [idx$b_keylen];
2270 3068 4         CH$MOVE(.parent_block + index$c_entries + .parent_block [index$w_used]
2271 3069 4           - (.parent_entry + .parent_entry_siz),
2272 3070 4           .parent_entry + .parent_entry_siz,
2273 3071 4           .parent_entry );      ! compress to cover old entry
2274 3072 4         parent_block [index$w_used] = .parent_block [index$w_used] - .parent_entry_siz;
2275 3073 4         perform (add_index2 (.index, .vbn, .index_block) );
2276 3074 3       END;
2277 3075 3     END
2278 3076 2   ELSE
2279 3077 2     RETURN lbr$_intrnlerr;      ! reset_highest2 only for ASCII keys
2280 3078 2
2281 3079 2      Mark the parent index block modified.
2282 3080 2
2283 3081 2 mark_dirty(.index_block [index$l_parent]);
2284 3082 2
2285 3083 2      Reset the highest key in the parent's parent.
```


reset_highest2

```
2286 3084 2 | Must check that .parent_block is the address of block .index_block [index$l_parent]
2287 3085 2 | since the last call to add index2 may have resulted in a new parent.
2288 3086 2 | If there is a new parent then it has already been reset.
2289 3087 2 |
2290 3088 3 BEGIN
2291 3089 3 LOCAL
2292 3090 3 blk_adr,
2293 3091 3 status;
2294 3092 3
2295 3093 3 perform ( find_index ( .index_block [index$l_parent], blk_adr ) );
2296 3094 3 IF .blk_adr EQ[ .parent_block
2297 3095 3 THEN
2298 3096 4 BEGIN
2299 3097 4 status = reset_highest2(.index, .index_desc, .index_block [index$l_parent], .parent_block);
2300 3098 4 IF NOT .status THEN RETURN lbr$_intrn[err;
2301 3099 3 END;
2302 3100 2 END;
2303 3101 2
2304 3102 2 RETURN true;
2305 3103 2
2306 3104 1 END;
```

OFFC 00000 RESET_HIGHEST2:

5E	08	C2	00002	WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	2971
57	10	AC	D0 00005	SUBL2	#8, SP	
	02	A7	D5 00009	MOVL	INDEX_BLOCK, R7	3012
		03	12 0000C	TSTL	2(R7)	
		00D2	31 0000E	BNEQ	1\$	
50	0C	A7	9E 00011	BRW	11\$	
51		67	3C 00015	MOVAB	12(R7), NEXT_ENTRY	3018
51	0C	A147	9E 00018	MOVZWL	(R7), R1	3019
51		50	D1 0001D	MOVAB	12(R1)[R7], R1	
		0E	18 00020	CMPL	NEXT_ENTRY, R1	
52		50	D0 00022	BGEQ	3\$	
51	06	A0	9A 00025	MOVL	NEXT_ENTRY, LAST_ENTRY	3021
50	07	A140	9E 00029	MOVZBL	6(NEXT_ENTRY), R1	3022
		E5	11 0002E	MOVAB	7(R1)[NEXT_ENTRY], NEXT_ENTRY	
51		6E	9E 00030	BRB	2\$	3019
50	02	A7	D0 00033	MOVAB	PARENT_BLOCK, R1	3027
		FD21	30 00037	MOVL	2(R7), R0	
6F		50	E9 0003A	BSBW	FIND_INDEX	
58		6E	D0 0003D	BLBC	STATUS, 8\$	
50	0C	A8	9E 00040	MOVL	PARENT_BLOCK, R8	3031
53	0200	C8	9E 00044	MOVAB	12(R8), ENTRY	
OC	AC	60	D1 00049	MOVAB	512(R8), R3	3045
		05	12 0004D	CMPL	(ENTRY), VBN	3037
56		50	D0 0004F	BNEQ	5\$	
		10	11 00052	MOVL	ENTRY, PARENT_ENTRY	3040
51	06	A0	9A 00054	BRB	6\$	3039
50	07	A140	9E 00058	MOVZBL	6(ENTRY), R1	3044
53		50	D1 0005D	MOVAB	7(R1)[ENTRY], ENTRY	
		E7	15 00060	CMPL	ENTRY, R3	3045
				BLEQ	4\$	

			77	11	00062	BRB	10\$	:	3046
		06	73	08	BC E9 00064	BLBC	@INDEX DESC, 10\$	:	3051
		06	A2	06	A6 91 00068	CMQB	6(PARENT_ENTRY), 6(LAST_ENTRY)	:	3054
					11 12 0006D	BNEQ	7\$	:	
		06	A6	06	A2 90 0006F	MOVB	6(LAST_ENTRY), 6(PARENT_ENTRY)	:	3057
			50	06	A2 9A 00074	MOVZBL	6(LAST_ENTRY), R0	:	3058
07	A6	07	A2		50 28 00078	MOVCL	R0, 7(LAST_ENTRY), 7(PARENT_ENTRY)	:	3060
					2F 11 0007E	BRB	9\$	:	3054
			59	06	A6 9A 00080	MOVZBL	6(PARENT_ENTRY), PARENT_ENTRY_SIZ	:	3067
			59		07 C0 00084	ADDL2	#7, PARENT_ENTRY_SIZ	:	
			50		68 3C 00087	MOVZWL	(R8), R0	:	3068
51			58		50 C1 0008A	ADDL3	R0, R8, R1	:	
50			56		59 C1 0008E	ADDL3	PARENT_ENTRY_SIZ, PARENT_ENTRY, R0	:	3069
			51		50 C2 00092	SUBL2	R0, R1	:	
			51		0C C0 00095	ADDL2	#12, R1	:	
66			60		51 28 00098	MOVCL	R1, (R0), (PARENT_ENTRY)	:	3071
			68		59 A2 0009C	SUBW2	PARENT_ENTRY_SIZ, (R8)	:	3072
					57 DD 0009F	PUSHL	R7	:	3073
				0C	AC DD 000A1	PUSHL	VBN	:	
				04	AC DD 000A4	PUSHL	INDEX	:	
FE6D		CF			03 FB 000A7	CALLS	#3, ADD_INDEX2	:	
		37			50 E9 000AC	BLBC	STATUS, -12\$	:	
		50		02	A7 D0 000AF	MOVL	2(R7), R0	:	3081
					0000V 30 000B3	BSBW	MARK DIRTY	:	
		51		04	AE 9E 000B6	MOVAB	BLK_ADR, R1	:	3093
		50		02	A7 D0 000BA	MOVL	2(R7), R0	:	
					FC9A 30 000BE	BSBW	FIND_INDEX	:	
		22			50 E9 000C1	BLBC	STATUS, 12\$	:	
		58		04	AE D1 000C4	CMPL	BLK_ADR, R8	:	3094
					19 12 000C8	BNEQ	11\$	:	
					58 DD 000CA	PUSHL	R8	:	3097
				02	A7 DD 000CC	PUSHL	2(R7)	:	
		7E		04	AC 7D 000CF	MOVQ	INDEX, -(SP)	:	
FF28		CF			04 FB 000D3	CALLS	#4, RESET_HIGHEST2	:	
		08			50 E8 000D8	BLBS	STATUS, 1T\$	:	3098
		50	00000000G		8F D0 000DB	MOVL	#LBR\$_INTRNLERR, R0	:	
					04 000E2	RET		:	
		50		01	D0 000E3	MOVL	#1, R0	:	3102
					04 000E6	RET		:	3104

; Routine Size: 231 bytes, Routine Base: \$CODE\$ + 0F34

check_lock

```

: 2308      3105 1 %SBTTL 'check_lock';
: 2309      3106 1 GLOBAL ROUTINE-check_lock : JSB_0 =
: 2310      3107 2 BEGIN
: 2311      3108 2
: 2312      3109 2 ---
: 2313      3110 2
: 2314      3111 2      Check if the index is locked from modification.
: 2315      3112 2
: 2316      3113 2      Inputs:
: 2317      3114 2
: 2318      3115 2      None
: 2319      3116 2
: 2320      3117 2      Outputs:
: 2321      3118 2
: 2322      3119 2      None
: 2323      3120 2
: 2324      3121 2      Routine value:
: 2325      3122 2
: 2326      3123 2      true      Ok to modify index
: 2327      3124 2      lbr$_updurtrav  Index is locked
: 2328      3125 2
: 2329      3126 2 ---
: 2330      3127 2
: 2331      3128 2 BIND
: 2332      3129 2      index_desc = .lbr$gl_control [lbr$_hdrptr] + lhd$_idxdesc      ! Name index descriptor for current
: 2333      3130 2      + (.lbr$gl_control [lbr$_curidx] - 1) * idd$_length
: 2334      3131 2      : BBLOCK;
: 2335      3132 2
: 2336      3133 2 IF .index_desc [idd$_locked]
: 2337      3134 2 THEN RETURN lbr$_updurtrav
: 2338      3135 2 ELSE RETURN true
: 2339      3136 2
: 2340      3137 1 END;

! Of check_lock
```

51	0000G	CF	D0 00000	CHECK_LOCK::		
50	12	A1	D0 00005	MOVL	LBR\$GL_CONTROL, R1	: 3129
50	0A B140	7E	00009	MOVL	18(R1), R0	: 3130
50	00BC	C0	9E 0000E	MOVAQ	@10(R1)(R0), R0	:
60		01	E1 00013	MOVAB	188(R0), R0	:
50	00000000G	8F	D0 00017	BBC	#1, (R0), 1\$	: 3133
			05 0001E	MOVL	#LBR\$_UPDURTRAV, R0	: 3135
50		01	D0 0001F 1\$:	RSB		:
			05 00022	MOVL	#1, R0	: 3137
				RSB		:

; Routine Size: 35 bytes, Routine Base: \$CODE\$ + 101B

mark_dirty

```
: 2342 3138 1 %SBTTL 'mark_dirty';
: 2343 3139 1 GLOBAL ROUTINE mark_dirty (vbn) : JSB_1 =
: 2344 3140 1
: 2345 3141 1 ---
: 2346 3142 1
: 2347 3143 1 Mark an index block modified in memory so that
: 2348 3144 1 it gets written back to disk when the file is closed.
: 2349 3145 1
: 2350 3146 1 Inputs:
: 2351 3147 1
: 2352 3148 1 vbn = disk block number
: 2353 3149 1
: 2354 3150 1 Outputs:
: 2355 3151 1
: 2356 3152 1 None
: 2357 3153 1 ---
: 2358 3154 1
: 2359 3155 2 BEGIN
: 2360 3156 2
: 2361 3157 2 LOCAL
: 2362 3158 2 cache_entry: REF BBLOCK;
: 2363 3159 2
: 2364 3160 2 perform (lookup_cache (.vbn, cache_entry)); ! Lookup entry in cache
: 2365 3161 2
: 2366 3162 2 cache_entry [cache$v_dirty] = true; ! Mark modified
: 2367 3163 2
: 2368 3164 2 RETURN true;
: 2369 3165 2
: 2370 3166 1 END;
```

5E	04	C2	00000	MARK_DIRTY::	SUBL2	#4, SP	: 3139
51	6E	9E	00003		MOVAB	CACHE_ENTRY, R1	: 3160
	0000G	30	00006		BSBW	LOOKUP_CACHE	
0A	50	E9	00009		BLBC	STATUS, 1\$	
50	6E	D0	0000C		MOVL	CACHE_ENTRY, R0	: 3162
OC	A0	01	88	0000F	BISB2	#1, 12(R0)	
50	01	D0	00013		MOVL	#1, R0	: 3164
5E	04	C0	00016	1\$:	ADDL2	#4, SP	: 3166
		05	00019		RSB		:

; Routine Size: 26 bytes, Routine Base: \$CODE\$ + 103E

```
: 2371 3167 1
: 2372 3168 1 END
: 2373 3169 0 ELUDOM
```

: F

PSECT SUMMARY

Name	Bytes	Attributes
\$CODE\$	4184	NOVEC,NOWRT, RD , EXE,NOSHR, LCL, REL, CON,NOPIC,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[SYSLIB]STARLET.L32;1	9776	14	0	581	00:01.0

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:INDEX/OBJ=OBJ\$:INDEX MSRC\$:INDEX/UPDATE=(ENH\$:INDEX)

Size: 4184 code + 0 data bytes

Run Time: 01:25.6

Elapsed Time: 02:52.4

Lines/CPU Min: 2220

Lexemes/CPU-Min: 22428

Memory Used: 377 pages

Compilation Complete

0198 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

GETHELP
LIS

INDEX
LIS

GETPUT
LIS

GETMEM
LIS

0199 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

OLDLIB
LIS

OPENCLOSE
LIS

OUTPUTLP
LIS

LBRMSG
LIS