



|          |    |      |        |        |      |           |    |    |          |          |           |
|----------|----|------|--------|--------|------|-----------|----|----|----------|----------|-----------|
| RRRRRRRR | MM | MM   | 333333 | NN     | NN   | EEEEEEEEE | XX | XX | TTTTTTTT | RRRRRRRR | EEEEEEEEE |
| RRRRRRRR | MM | MM   | 333333 | NN     | NN   | EEEEEEEEE | XX | XX | TTTTTTTT | RRRRRRRR | EEEEEEEEE |
| RR       | RR | MMMM | MMMM   | 33     | NN   | EE        | XX | XX | TT       | RR       | RR        |
| RR       | RR | MMMM | MMMM   | 33     | NN   | EE        | XX | XX | TT       | RR       | RR        |
| RR       | RR | MM   | MM     | 33     | NNNN | NN        | EE | XX | XX       | RR       | RR        |
| RR       | RR | MM   | MM     | 33     | NNNN | NN        | EE | XX | XX       | RR       | RR        |
| RRRRRRRR | MM | MM   | 33     | NN     | NN   | EEEEEEE   | XX | XX | TT       | RRRRRRRR | EEEEEEE   |
| RRRRRRRR | MM | MM   | 33     | NN     | NN   | EEEEEEE   | XX | XX | TT       | RRRRRRRR | EEEEEEE   |
| RR       | RR | MM   | MM     | 33     | NN   | NNNN      | EE | XX | XX       | RR       | RR        |
| RR       | RR | MM   | MM     | 33     | NN   | NNNN      | EE | XX | XX       | RR       | RR        |
| RR       | RR | MM   | MM     | 33     | NN   | EE        | XX | XX | TT       | RR       | RR        |
| RR       | RR | MM   | MM     | 33     | NN   | EE        | XX | XX | TT       | RR       | RR        |
| RR       | RR | MM   | MM     | 333333 | NN   | EEEEEEEEE | XX | XX | TT       | RR       | RR        |
| RR       | RR | MM   | MM     | 333333 | NN   | EEEEEEEEE | XX | XX | TT       | RR       | RR        |

|            |  |         |
|------------|--|---------|
| LL         |  | SSSSSSS |
| LL         |  | SSSSSSS |
| LL         |  | SS      |
| LLLLLLLLLL |  | SSSSSSS |
| LLLLLLLLLL |  | SSSSSSS |

```
1 0001 0 MODULE RM3NEXTRE (LANGUAGE (BLISS32) .
2 0002 0 IDENT = 'V04-000'
3 0003 0 )
4 0004 0
5 0005 1 BEGIN
6 0006 1 ****
7 0007 1 *
8 0008 1 *
9 0009 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
10 0010 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
11 0011 1 * ALL RIGHTS RESERVED.
12 0012 1 *
13 0013 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
14 0014 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
15 0015 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
16 0016 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
17 0017 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
18 0018 1 * TRANSFERRED.
19 0019 1 *
20 0020 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
21 0021 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
22 0022 1 * CORPORATION.
23 0023 1 *
24 0024 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
25 0025 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
26 0026 1 *
27 0027 1 *
28 0028 1 ****
29 0029 1
30 0030 1 ++
31 0031 1
32 0032 1 FACILITY: RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
33 0033 1
34 0034 1 ABSTRACT:
35 0035 1 This module contains routines to skip records and skip
36 0036 1 overhead for key compares
37 0037 1
38 0038 1
39 0039 1 ENVIRONMENT.
40 0040 1
41 0041 1 VAX/VMS OPERATING SYSTEM
42 0042 1
43 0043 1 --
44 0044 1
45 0045 1
46 0046 1 AUTHOR: Todd M. Katz RECREATION DATE: 05-Sep-1982
47 0047 1
48 0048 1 Modified by:
49 0049 1
50 0050 1 V03-011 DGB0001 Donald G. Blair 05-Dec-1983
51 0051 1 The routine RMS$REC_OVHD was found to be critical to performance.
52 0052 1 Therefore, I've rewritten/reorganized the routine to make it
53 0053 1 faster, somewhat at the expense of ease of understanding the
54 0054 1 code.
55 0055 1
56 0056 1 V03-010 MCN0002 Maria del C. Nasr 22-Mar-1983
57 0057 1 More linkages changes, in particular, RMS$REC_OVHD.
```

```
58 0058 1
59 0059 1 V03-009 MCN0001 Maria del C. Nasr 24-Feb-1983
60 0060 1 Reorganize linkages
61 0061 1
62 0062 1 V03-008 TMK0006 Todd M. Katz 28-Sep-1982
63 0063 1 Fix an error in RMS$REC_OVHD. For prologue 3 files, all types
64 0064 1 of primary data records have a record size field except for
65 0065 1 the fixed length formatted records of those files that have
66 0066 1 both key compression and data compression turned off. I was not
67 0067 1 checking the compression bits; but rather, just that the record
68 0068 1 format was fixed before treated the records as if they did not
69 0069 1 have a record size field. Of course, this caused problems
70 0070 1 because both the record overhead and the size of the record
71 0071 1 would be incorrectly determined.
72 0072 1
73 0073 1 V03-007 TMK0005 Todd M. Katz 05-Sep-1982
74 0074 1 Re-write the remaining routines in this module (RMS$REC_OVHD,
75 0075 1 RMS$GETNEXT_REC, RMS$COMPARE_REC), and check in a new source
76 0076 1 since all the routines in this module will have been newly
77 0077 1 added or re-written. While doing this, add support for Prologue
78 0078 1 3 SIDRs to those routines which require modification.
79 0079 1
80 0080 1 ***
81 0081 1
82 0082 1 LIBRARY 'RMSLIB:RMS';
83 0083 1
84 0084 1 REQUIRE 'RMSSRC:RMSIDXDEF';
85 0149 1
86 0150 1 ! Define default PSECTS for code
87 0151 1
88 0152 1 PSECT
89 0153 1 CODE = RMS$RMS3(PSECT_ATTR),
90 0154 1 PLIT = RMS$RMS3(PSECT_ATTR);
91 0155 1
92 0156 1 ! Linkage
93 0157 1
94 0158 1 LINKAGE
95 0159 1 L_COMPARE_KEY,
96 0160 1 L_PRESERVE1,
97 0161 1 L_RABREG 67,
98 0162 1 L_REC_OVHD,
99 0163 1 L_SIDR_FIRST;
100 0164 1
101 0165 1 ! External routines
102 0166 1
103 0167 1 EXTERNAL ROUTINE
104 0168 1 RMS$COMPARE_KEY : RL$COMPARE_KEY,
105 0169 1 RMS$RECORD_KEY : RL$PRESERVE1,
106 0170 1 RMS$RECORD_VBN : RL$PRESERVE1;
107 0171 1
108 0172 1 ! Forward Routine
109 0173 1
110 0174 1 FORWARD ROUTINE RMS$REC_OVHD : RL$REC_OVHD;
```

```
113      0176 1 %SBTTL 'RMSCOMPARE REC'  
114      0177 1 GLOBAL ROUTINE RMSCOMPARE_REC (SRCH_KEY_ADDR, SRCH_KEY_SIZE, LEVEL) :  
115          0178 1 RL$RABREG_67 =  
116          0179 1 !++  
117          0180 1  
118          0181 1  
119          0182 1 FUNCTIONAL DESCRIPTION:  
120          0183 1  
121          0184 1 This routine compares a key within a primary data or SIDR record with  
122          0185 1 a search key. There are three possible comparisons that can take place  
123          0186 1 depending upon the key of reference of the index descriptor and the  
124          0187 1 level of the bucket containing the record as represented by the input  
125          0188 1 parameter LEVEL:  
126          0189 1  
127          0190 1 1. Search Key vs. Primary Key in Primary Data Record.  
128          0191 1 In this case either both the key of reference and LEVEL are 0, or  
129          0192 1 the key of reference is 0, and LEVEL is -1.  
130          0193 1  
131          0194 1 2. Search Key vs. Secondary Key in Primary Data Record.  
132          0195 1 In this case the key of reference will be other than the primary key,  
133          0196 1 and LEVEL will be -1 indicating that the record is a primary data  
134          0197 1 record.  
135          0198 1  
136          0199 1 3. Search Key vs. Secondary Key in Secondary Data Record (SIDR).  
137          0200 1 In this case the key of reference will be other than the primary key,  
138          0201 1 and LEVEL will be 0. There is of course, only one key in such a  
139          0202 1 record.  
140          0203 1  
141          0204 1 One special note: A LEVEL of -1 indicates that RMS will have to compare  
142          0205 1 the search key with some key in this primary data record, but which key  
143          0206 1 depends upon the key of reference of the input index descriptor. If the  
144          0207 1 file is a prologue 3 file, then the primary key will "look" different  
145          0208 1 from the alternate keys since it will be at the front of the record, and  
146          0209 1 might also be compressed. What is more important is that because  
147          0210 1 the primary key has been extracted from the rest of the primary data  
148          0211 1 record, and the primary data record itself might be compressed, if the  
149          0212 1 comparison is to be made between the search key and a secondary key in  
150          0213 1 the primary data record, it will be impossible to find the secondary  
151          0214 1 key in the record. In fact, it will be impossible to find any secondary  
152          0215 1 key in the record. Therefore, If the file is a prologue 3 file, and the  
153          0216 1 LEVEL is -1, this routine requires that REC_ADDR point to an unpacked  
154          0217 1 version of the primary data record.  
155          0218 1  
156          0219 1 CALLING SEQUENCE:  
157          0220 1  
158          0221 1 RMSCOMPARE_REC ()  
159          0222 1  
160          0223 1 INPUT PARAMETERS:  
161          0224 1  
162          0225 1 SRCH_KEY_ADDR - address of the search key  
163          0226 1 SRCH_KEY_SIZE - size of the search key  
164          0227 1 LEVEL - if 0, then the record is primary data record or SIDR  
165          0228 1 if -1, then the record is a primary data record  
166          0229 1  
167          0230 1 IMPLICIT INPUTS:  
168          0231 1  
169          0232 1      IDX_DFN - address of index descriptor
```

```

170      0233 1 |     IDX$V_KEY_COMPR      - if set, key compression is enabled
171      0234 1 |     IDX$B_KEYREF      - key of reference
172      0235 1 |
173      0236 1 |     IFAB             - address of IFAB
174      0237 1 |     IFB$W_KBUFSZ    - size of a keybuffer
175      0238 1 |     IFB$B_PLG_VER   - prologue version of file
176      0239 1 |
177      0240 1 |     IRAB             - address of IRAB
178      0241 1 |     IRB$L_KEYBUF    - address of the contiguous keybuffers
179      0242 1 |
180      0243 1 |     REC_ADDR         - address of data record
181      0244 1 |
182      0245 1 |     OUTPUT PARAMETERS:
183      0246 1 |           NONE
184      0247 1 |
185      0248 1 |     IMPLICIT OUTPUTS:
186      0249 1 |           NONE
187      0250 1 |
188      0251 1 |     ROUTINE VALUE:
189      0252 1 |           -1      - search key < key of data record
190      0253 1 |           0       - search key = key of data record
191      0254 1 |           1       - search key > key of data record
192      0255 1 |
193      0256 1 |
194      0257 1 |     SIDE EFFECTS:
195      0258 1 |
196      0259 1 |           AP will be trashed.
197      0260 1 |           If key compression is enabled, and LEVEL = 0, then the key of the data
198      0261 1 |           record maybe found in its expanded form in keybuffer 5.
199      0262 1 |
200      0263 1 |---|
201      0264 1 |     BEGIN
202      0265 2 |     BUILTIN
203      0266 2 |           AP;
204      0267 2 |
205      0268 2 |
206      0269 2 |
207      0270 2 |     EXTERNAL REGISTER
208      0271 2 |           R_IDX_DFN_STR,
209      0272 2 |           R_IFAB_STR,
210      0273 2 |           R_IRAB_STR,
211      0274 2 |           R_REC_ADDR;
212      0275 2 |
213      0276 2 |
214      0277 2 |     LOCAL
215      0278 2 |           COMPARE_KEY;
216      0279 2 |
217      0280 2 |           Either the search key is to be compared with the primary key of the
218      0281 2 |           the primary data record or the secondary (and only) key of the SIDR.
219      0282 2 |           IF .LEVEL EQLU 0
220      0283 2 |           THEN
221      0284 2 |
222      0285 2 |           If key compression is enabled, then the primary or secondary key of
223      0286 2 |           the data record must be extracted and re-expanded before it can be
224      0287 2 |           compared with the search key. A contiguous key - contiguous key
225      0288 2 |           comparison will be made between the search key, and the key of the
226      0289 2 |           data record in its expanded form in keybuffer 5.

```

```
227      0290 2
228      0291 2
229      0292 2
230      0293 2
231      0294 2
232      0295 2
233      0296 2
234      0297 2
235      0298 2
236      0299 2
237      0300 2
238      0301 2
239      0302 2
240      0303 2
241      0304 2
242      0305 2
243      0306 2
244      0307 2
245      0308 2
246      0309 2
247      0310 2
248      0311 2
249      0312 2
250      0313 2
251      0314 2
252      0315 2
253      0316 2
254      0317 2
255      0318 2
256      0319 2
257      0320 2
258      0321 2
259      0322 2
260      0323 2
261      0324 3
262      0325 3
263      0326 4
264      0327 4
265      0328 4
266      0329 4
267      0330 4
268      0331 4
269      0332 4
270      0333 4
271      0334 4
272      0335 4
273      0336 4
274      0337 4
275      0338 4
276      0339 4
277      0340 4
278      0341 4
279      0342 4
280      0343 4
281      0344 4
282      0345 4
283      0346 4

      ! IF .IDX_DFN[IDX$V_KEY_COMPR]
      THEN
        BEGIN
          GLOBAL REGISTER
            R_BDB,
            R_RAB,
            R_IMPURE;

          AP = 0;
          RMSRECORD_KEY (KEYBUF_ADDR(5));

          AP = 3;
          COMPARE_KEY = KEYBUF_ADDR(5);
        END

      ! Key compression is not enabled, therefore there is no need to
      re-expand the key of the data record.

      ELSE
        BEGIN
          AP = 3;

          ! The comparison will be made between the search key and the primary
          key of a primary data record. If the file is a prologue 3 file,
          then RMS will position directly to the contiguous primary key in
          the primary data record before the comparison, and a contiguous
          key - contiguous key comparison will be performed; otherwise, RMS
          will position past the record overhead directly to the primary
          data record before the comparison is made, and a contiguous
          search key - data record comparison is performed.

        IF .IDX_DFN[IDX$B_KEYREF] EQLU 0
        THEN
          BEGIN
            IF .IFBSB_PLG_VER LSSU PLG$C_VER_3
            THEN
              AP = 2;
            END

          ! The comparison will be made between the search key and the
          secondary key in the SIDR. Regardless of the prologue version
          of the file, RMS must position past the record overhead to the
          secondary key itself, and perform a contiguous key - contiguous
          key comparison.

        ELSE
          LEVEL = -1;

          ! Position past the record overhead either directly to the key
          the search key is to be compared with, or to the beginning of the
          primary data record proper.

          COMPARE_KEY = .REC_ADDR + RMSREC_OVHD(.LEVEL);
        END
```

```

: 284      0347 3
: 285      0348 3
: 286      0349 3
: 287      0350 3
: 288      0351 3
: 289      0352 2
: 290      0353 2
: 291      0354 2
: 292      0355 2
: 293      0356 3
: 294      0357 3
: 295      0358 3
: 296      0359 3
: 297      0360 3
: 298      0361 3
: 299      0362 3
: 300      0363 4
: 301      0364 4
: 302      0365 4
: 303      0366 4
: 304      0367 4
: 305      0368 4
: 306      0369 4
: 307      0370 4
: 308      0371 4
: 309      0372 3
: 310      0373 3
: 311      0374 2
: 312      0375 2
: 313      0376 2
: 314      0377 2
: 315      0378 2
: 316      0379 1

| The search key will be compared with a key in a primary data record.  

| Which key it is compared with will depend upon the key of reference of the  

| index descriptor.

ELSE
  BEGIN
    AP = 2;

  | If the file is a prologue 2 file, then it will be necessary to
  | position past the record overhead (and do so) before the search key
  | is compared with the a key in the primary data record by means of a
  | contiguous search key - data record comparison.

  IF .IFAB[IFBSB_PLG_VER] LSSU PLG$C_VER_3
  THEN
    BEGIN
      COMPARE_KEY = .REC_ADDR + RM$REC_OVHD(0);
    END

  | If the file is a prologue 3 file, then REC_ADDR points directly to an
  | unpacked version of the primary data record. There will be no need to
  | perform any initial positioning, and the type of comparison which
  | will be required will be contiguous search key - data record.

  ELSE
    COMPARE_KEY = .REC_ADDR;
  END;

| Perform the required comparison, and return the result.

RETURN RMSCOMPARE_KEY (.COMPARE_KEY, .SRCH_KEY_ADDR, .SRCH_KEY_SIZE);
END;

```

```

.TITLE RM3NEXTRE
.IDENT \V04-000\

.EXTRN RMSCOMPARE_KEY, RM$RECORD_KEY
.EXTRN RMSRECORD_VBN

.PSECT RMSRMS3,NOWRT, GBL, PIC,2

```

|    | 0918  | 8F          | BB 00000 RMSCOMPARE REC::        |  |      |
|----|-------|-------------|----------------------------------|--|------|
|    |       |             | PUSHR #^M<R3,R4,R8,R11>          |  | 0177 |
|    |       |             | TSTL LEVEL                       |  | 0282 |
| 20 | 1C A7 |             | BNEQ 4\$                         |  |      |
|    |       |             | BBC #6, 28(IDX_DFN), 1\$         |  | 0291 |
|    |       |             | CLRL AP                          |  | 0300 |
|    | 50    | 00B4        | MOVZWL 180(IFAB), R0             |  | 0301 |
|    |       | 60 B940     | PUSHAL @96(IRAB)[R0]             |  |      |
|    |       | 0000G       | BSBW RMSRECORD_KEY               |  |      |
|    |       | 30 00019    | ADDL2 #4, SP                     |  |      |
|    | 5E    | 04          | MOVL #3, AP                      |  | 0303 |
|    | 5C    | 03          | MOVZWL 180(IFAB), R0             |  | 0304 |
|    | 50    | 00B4        | MOVAL @96(IRAB)[R0], COMPARE_KEY |  |      |
|    | 54    | 60 B940     | BRB 7\$                          |  | 0291 |
|    |       | 36 11 0002C |                                  |  |      |

|    |    |      |      |       |    |       |      |       |                           |      |
|----|----|------|------|-------|----|-------|------|-------|---------------------------|------|
|    |    | 5C   | 21   | 03    | D0 | 0002E | 1\$: | MOVL  | #3, AP                    | 0313 |
|    |    |      |      | A7    | 95 | 00031 |      | TSTB  | 33(IDX_DFN)               | 0324 |
|    |    | 03   | 00B7 | 0C    | 12 | 00034 |      | BNEQ  | 2\$                       |      |
|    |    |      |      | CA    | 91 | 00036 |      | CMPB  | 183(IFAB), #3             | 0327 |
|    |    |      |      | 09    | 1E | 00038 |      | BGEQU | 3\$                       |      |
|    |    | 5C   |      | 02    | D0 | 0003D |      | MOVL  | #2, AP                    | 0329 |
|    |    |      |      | 04    | 11 | 00040 |      | BRB   | 3\$                       | 0324 |
|    | 1C | AE   | 1C   | 01    | CE | 00042 | 2\$: | MNEGL | #1, LEVEL                 | 0339 |
|    |    | 51   |      | AE    | D0 | 00046 | 3\$: | MOVL  | LEVEL, R1                 | 0345 |
|    |    | 5C   |      | 0C    | 11 | 0004A |      | BRB   | 5\$                       |      |
|    |    | 03   | 00B7 | 02    | D0 | 0004C | 4\$: | MOVL  | #2, AP                    | 0354 |
|    |    |      |      | CA    | 91 | 0004F |      | CMPB  | 183(IFAB), #3             | 0361 |
|    |    |      |      | 0B    | 1E | 00054 |      | BGEQU | 6\$                       |      |
|    |    |      |      | 51    | D4 | 00056 |      | CLRL  | R1                        | 0364 |
|    |    |      |      | 0000V | 30 | 00058 | 5\$: | BSBW  | RMSREC OVHD               |      |
| 54 |    | 50   |      | 56    | C1 | 0005B |      | ADDL3 | REC_ADDR, R0, COMPARE_KEY |      |
|    |    |      |      | 03    | 11 | 0005F |      | BRB   | 7\$                       | 0361 |
|    |    | 54   |      | 56    | D0 | 00061 | 6\$: | MOVL  | REC_ADDR, COMPARE_KEY     | 0373 |
|    |    | 50   | 18   | AE    | D0 | 00064 | 7\$: | MOVL  | SRCH_KEY_SIZE, R0         | 0378 |
|    |    | 53   | 14   | AE    | D0 | 00068 |      | MOVL  | SRCH_KEY_ADDR, R3         |      |
|    |    | 51   |      | 54    | D0 | 0006C |      | MOVL  | COMPARE KEY, R1           |      |
|    |    |      |      | 0000G | 30 | 0006F |      | BSBW  | RMSCOMPARE KEY            |      |
|    |    | 0918 |      | 8F    | BA | 00072 |      | POPR  | #^M<R3,R4,R8,R11>         | 0379 |
|    |    |      |      |       | 05 | 00076 |      | RSB   |                           |      |

: Routine Size: 119 bytes. Routine Base: RMSRMS3 + 0000

: 318 0380 1 %SBTTL 'RMSEXT\_ARRY\_RFA'  
319 0381 1 GLOBAL ROUTINE RMSEXT\_ARRY\_RFA (VBN, ID) : RL\$RABREG\_67 =  
320 0382 1  
321 0383 1 ++  
322 0384 1  
323 0385 1 FUNCTIONAL DESCRIPTION:  
324 0386 1  
325 0387 1 This routine's responsibility is to extract out and return the  
326 0388 1 components of the current SIDR array element's RFA pointer, provided  
327 0389 1 the current element has not been flattened, and there is a RFA  
328 0390 1 pointer to extract.  
329 0391 1  
330 0392 1 CALLING SEQUENCE:  
331 0393 1 BSBW RMSEXT\_ARRY\_RFA()  
332 0394 1  
333 0395 1 INPUT PARAMETERS:  
334 0396 1 NONE  
335 0397 1  
336 0398 1 IMPLICIT INPUTS:  
337 0399 1  
338 0400 1 IFAB - address of the IFAB  
339 0401 1 IFBSB\_PLG\_VER - prologue version of the ISAM file  
340 0402 1  
341 0403 1 REC\_ADDR - address of the current SIDR array element  
342 0404 1  
343 0405 1 OUTPUT PARAMETERS:  
344 0406 1  
345 0407 1 VBN - VBN of the current SIDR array element's RFA pointer  
346 0408 1 ID - ID of the current SIDR array element's RFA pointer  
347 0409 1  
348 0410 1 IMPLICIT OUTPUTS:  
349 0411 1 NONE  
350 0412 1  
351 0413 1 ROUTINE VALUE:  
352 0414 1  
353 0415 1 SUC - if the RFA has been successfully extracted.  
354 0416 1 DEL - if the current SIDR array element is marked deleted.  
355 0417 1 0 - if the cuurent SIDR array element is flattened  
356 0418 1 (ie no RFA pointer is present).  
357 0419 1  
358 0420 1 SIDE EFFECTS:  
359 0421 1 If SUC is returned, the SIDR array's RFA is returned too.  
360 0422 1 If DEL is returned, the SIDR array's RFA is returned too.  
361 0423 1 If 0 is returned, the SIDR array's RFA is not returned.  
362 0424 1 AP is trashed.  
363 0425 1  
364 0426 1 ---  
365 0427 1  
366 0428 2 BEGIN  
367 0429 2  
368 0430 2 BUILTIN  
369 0431 2 AP;  
370 0432 2  
371 0433 2 EXTERNAL REGISTER  
372 0434 2 COMMON RAB\_STR,  
373 0435 2 R\_IDX\_BFN\_STR,  
374 0436 2 R\_REC\_ADDR\_STR;

```

375 0437 2
376 0438 2 GLOBAL REGISTER
377 0439 2 R_BDB;
378 0440 2
379 0441 2 | If there is no SIDR array RFA to extract, return 0.
380 0442 2
381 0443 2 IF .REC_ADDR[IRC$V_NOPTRSZ]
382 0444 2 THEN
383 0445 2 RETURN 0;
384 0446 2
385 0447 2 Extract the VBN and ID from the RFA pointer of the CURRENT SIDR element.
386 0448 2
387 0449 2 AP = 2;
388 0450 2 .VBN = RMSRECORD_VBN();
389 0451 2
390 0452 2 IF .IFAB[IFBSB_PLG_VER] GEQU PLG$C_VER_3
391 0453 2 THEN
392 0454 2 .ID = .REC_ADDR[IRC$W_ID]
393 0455 2 ELSE
394 0456 2 .ID = .REC_ADDR[IRC$B_ID];
395 0457 2
396 0458 2 | If the current SIDR array element is marked deleted, return that status.
397 0459 2
398 0460 2 IF .REC_ADDR[IRC$V_DELETED]
399 0461 2 THEN
400 0462 3 RETURN RMSERR(DEL)
401 0463 3
402 0464 3 | Otherwise, return success.
403 0465 3
404 0466 2 ELSE
405 0467 2 RETURN RMSSUC();
406 0468 2
407 0469 1 END;

```

| 54 DD 00000 RM\$EXT_ARRY RFA:: |    |      |       |       |               |        |                     |  |      |
|--------------------------------|----|------|-------|-------|---------------|--------|---------------------|--|------|
| 2D                             |    | 66   | 04    | E0    | 00002         | PUSHL  | R4                  |  | 0381 |
|                                |    | 5C   | 02    | D0    | 00006         | BBS    | #4, (REC_ADDR), 4\$ |  | 0443 |
|                                |    |      | 0000G | 30    | 00009         | MOVL   | #2, AP              |  | 0449 |
|                                | 08 | BE   | 50    | D0    | 0000C         | BSBW   | RMSRECORD_VBN       |  | 0450 |
|                                | 03 | 00B7 | CA    | 91    | 00010         | MOVL   | R0, @VBN            |  | 0452 |
|                                |    |      | 07    | 1F    | 00015         | CMPB   | 183(IFAB), #3       |  | 0454 |
|                                | 0C | BE   | 01    | A6    | 3C 00017      | BLSSU  | 1\$                 |  | 0454 |
|                                |    |      | 05    | 11    | 0001C         | MOVZWL | 1(REC_ADDR), @ID    |  | 0456 |
|                                | 0C | BE   | 01    | A6    | 9A 0001E 1\$: | BRB    | 2\$                 |  | 0460 |
|                                | 66 |      | 02    | E1    | 00023 2\$:    | MOVZBL | 1(REC_ADDR), @ID    |  | 0467 |
| 07                             | 50 | 8262 | 8F    | 3C    | 00027         | BBC    | #2, (REC_ADDR), 3\$ |  |      |
|                                |    |      | 07    | 11    | 0002C         | MOVZWL | #33378, R0          |  |      |
|                                | 50 |      | 01    | D0    | 0002E 3\$:    | BRB    | 5\$                 |  |      |
|                                |    |      | 02    | 11    | 00031         | MOVL   | #1, R0              |  |      |
|                                | 50 |      | 50    | D4    | 00033 4\$:    | BRB    | 5\$                 |  |      |
|                                |    |      | 10    | BA    | 00035 5\$:    | CLRL   | R0                  |  |      |
|                                |    |      | 05    | 00037 |               | POPR   | #^M<R4>             |  | 0469 |
|                                |    |      |       |       |               | RSB    |                     |  |      |

RM3NEXTRE  
V04-000

RMSEXT\_ARRY\_RFA

L 7  
16-Sep-1984 01:53:40    VAX-11 Bliss-32 V4.0-742  
14-Sep-1984 13:01:30    [RMS.SRC]RM3NEXTRE.B32;1

Page 10  
(3)

: Routine Size: 56 bytes.    Routine Base: RMS\$RMS3 + 0077

RM3  
V04

: R

409  
410 0470 1 %SBTTL 'RMSGETNEXT\_REC'  
411 0471 1 GLOBAL ROUTINE RMSGETNEXT\_REC : RL\$RABREG\_67 NOVALUE =  
412 0472 1 !++  
413 0473 1 FUNCTIONAL DESCRIPTION:  
414 0474 1 This routine accepts as input the address of a current index, primary  
415 0477 1 data, or SIDR record. It positions past this current record to a new  
416 0478 1 current record, the next record in the bucket. If there is no next  
417 0479 1 record, RMS positions to the first byte past the current record.  
418 0480 1  
419 0481 1 CALLING SEQUENCE:  
420 0482 1 RMSGETNEXT\_REC()  
421 0483 1  
422 0484 1 INPUT PARAMETERS:  
423 0485 1 NONE  
424 0486 1  
425 0487 1 IMPLICIT INPUTS:  
426 0488 1  
427 0489 1 0490 1  
428 0491 1 IDX\_DFN - address of index descriptor  
429 0492 1 IDX\$B\_KEYREF - key of reference  
430 0493 1  
431 0494 1 IFAB - address of IFAB  
432 0495 1 IFB\$B\_PLG\_VER - prologue version of file  
433 0496 1  
434 0497 1 IRAB - address of IRAB  
435 0498 1 IRBSL\_CURDBB - address of BDB for current record's bucket  
436 0499 1 IRBSL\_REC\_COUNT - number of preceding index records  
437 0500 1  
438 0501 1 REC\_ADDR - address of current record  
439 0502 1  
440 0503 1 OUTPUT PARAMETERS:  
441 0504 1 NONE  
442 0505 1  
443 0506 1 IMPLICIT OUTPUTS:  
444 0507 1  
445 0508 1 IRAB - address of IRAB  
446 0509 1 IRBSL\_REC\_COUNT - number of preceding index records  
447 0510 1  
448 0511 1 REC\_ADDR - address of the new record  
449 0512 1  
450 0513 1 ROUTINE VALUE:  
451 0514 1 NONE  
452 0515 1  
453 0516 1 SIDE EFFECTS:  
454 0517 1  
455 0518 1 If the current record is a prologue 3 index record, then the count  
456 0519 1 of the number of records preceding the current record is incremented  
457 0520 1 (there is a new current record, the next one in the bucket).  
458 0521 1  
459 0522 1 !--  
460 0523 1  
461 0524 2 BEGIN  
462 0525 2 EXTERNAL REGISTER  
463 0526 2

```

466      0527  2      R_IDX_DFN_STR,
467      0528  2      R_IFAB_STR,
468      0529  2      R_IRAB_STR,
469      0530  2      R_REC_ADDR_STR;
470      0531  2
471      0532  2      LOCAL
472      0533  2      REC_SIZE;
473      0534  2
474      0535  2      | Set the global register REC_SIZE according to whether the current record
475      0536  2      | is a:
476      0537  2
477      0538  2      | 1. Primary data record      -      REC_SIZE <- 0
478      0539  2      | 2. SIDR                      -      REC_SIZE <- -1
479      0540  2      | 3. Index record             -      REC_SIZE <- level of index bucket
480      0541  2
481      0542  2      REC_SIZE = .BBLOCK[.BBLOCK[.IRAB[IRB$L_CURBDB], BDB$L_ADDR], BKT$B_LEVEL];
482      0543  2
483      0544  3      IF (.REC_SIZE EQLU 0)
484      0545  2      AND
485      0546  3      (.IDX_DFN[IDXSB_KEYREF] GTRU 0)
486      0547  2      THEN
487      0548  2      REC_SIZE = -1;
488      0549  2
489      0550  2      | If the current record is a prologue 3 index record, then along with
490      0551  2      | positioning past it to what hopefully is the next record in the bucket,
491      0552  2      | RMS increments a count of the number of index records preceding the
492      0553  2      | new current record in the index bucket,
493      0554  2
494      0555  3      IF (.REC_SIZE GTRU 0)
495      0556  2      AND
496      0557  3      (.IFAB[IFBSB_PLG_VER] GEQU PLGSC_VER_3)
497      0558  2      THEN
498      0559  2      IRAB[IRB$L_REC_COUNT] = .IRAB[IRB$L_REC_COUNT] + 1;
499      0560  2
500      0561  2      | Position past the current record either to next record, or to the first
501      0562  2      | byte past the current record, if there is no next record.
502      0563  2
503      0564  2      REC_ADDR = .REC_ADDR + RMSREC OVHD(.REC_SIZE; REC_SIZE);
504      0565  2      REC_ADDR = .REC_ADDR + .REC_SIZE;
505      0566  1      END;

```

|    |      |                              |        |                  |  |      |
|----|------|------------------------------|--------|------------------|--|------|
| 50 | 20   | A9 D0 00000 RMSGETNEXT REC:: |        |                  |  |      |
| 50 | 18   | A0 D0 00004                  | MOVL   | 32(IRAB), R0     |  | 0542 |
| 51 | 0C   | A0 9A 00008                  | MOVL   | 24(R0), R0       |  |      |
|    | 08   | 12 0000C                     | MOVZBL | 12(R0), REC_SIZE |  |      |
|    | 21   | A7 95 0000E                  | BNEQ   | 1\$              |  | 0544 |
|    |      | 03 13 00011                  | TSTB   | 33(IPX_DFN)      |  | 0546 |
| 51 | 01   | CE 00013                     | BEQL   | 1\$              |  |      |
|    | 51   | D5 00016 1\$:                | MNEG.L | #1, REC_SIZE     |  | 0548 |
|    | 0B   | 13 00018                     | TSTL   | REC_SIZE         |  | 0555 |
| 03 | 00B7 | CA 91 0001A                  | BEQL   | 2\$              |  |      |
|    |      | 04 1F 0001F                  | CMPB   | 183(IFAB), #3    |  | 0557 |
|    |      |                              | BLSSU  | 2\$              |  |      |

RM3NEXTRE  
V04-000

RMSGETNEXT\_REC

B 8  
16-Sep-1984 01:53:40 VAX-11 Bliss-32 v4.0-742  
14-Sep-1984 13:01:30 [RMS.SRC]RM3NEXTRE.B32;1

Page 13  
(4)

|      |       |       |       |       |                    |      |
|------|-------|-------|-------|-------|--------------------|------|
| 0094 | C9    | D6    | 00021 | INCL  | 148(IRAB)          | 0559 |
| 56   | 0000V | 30    | 00025 | BSBW  | RMSREC OVHD        | 0564 |
|      | 50    | C0    | 00028 | ADDL2 | RO, REC_ADDR       |      |
| 56   | 51    | C0    | 0002B | ADDL2 | REC_SIZE, REC_ADDR | 0565 |
|      | 05    | 0002E |       | RSB   |                    | 0566 |

; Routine Size: 47 bytes, Routine Base: RM\$RMS3 + 00AF

RM3  
V04

S  
RELLMC

```
507      0567 1 %SBTTL 'RMSGETNXT ARRAY'  
508      0568 1 GLOBAL ROUTINE RMSGETNXT_ARRAY : RL$RABREG_67 NOVALUE =  
509      0569 1  
510      0570 1 !++  
511      0571 1  
512      0572 1 FUNCTIONAL DESCRIPTION:  
513      0573 1  
514      0574 1 This routine's purpose, is given a pointer to a current SIDR array  
515      0575 1 element, position to the SIDR array element that follows, or to the  
516      0576 1 first byte past the current element if there is no next element  
517      0577 1  
518      0578 1 CALLING SEQUENCE:  
519      0579 1 BSBW RMSGETNXT_ARRAY()  
520      0580 1  
521      0581 1 INPUT PARAMETERS:  
522      0582 1 NONE  
523      0583 1  
524      0584 1 IMPLICIT INPUTS:  
525      0585 1  
526      0586 1 IFAB - address of IFAB  
527      0587 1 IFBSB_PLG_VER - prologue version of the ISAM file  
528      0588 1  
529      0589 1 REC_ADDR - address of the current SIDR array element  
530      0590 1  
531      0591 1 OUTPUT PARAMETERS:  
532      0592 1 NONE  
533      0593 1  
534      0594 1 IMPLICIT OUTPUTS:  
535      0595 1 REC_ADDR - address of the next SIDR array element  
536      0596 1  
537      0597 1 ROUTINE VALUE:  
538      0598 1 NONE  
539      0599 1  
540      0600 1 SIDE EFFECTS:  
541      0601 1 NONE  
542      0602 1  
543      0603 1 !--  
544      0604 1  
545      0605 2 BEGIN  
546      0606 2  
547      0607 2 EXTERNAL REGISTER  
548      0608 2 R_IFAB_STR,  
549      0609 2 R_REC_ADDR_STR;  
550      0610 2  
551      0611 2 ! The current SIDR array element consists of just 1 control byte.  
552      0612 2  
553      0613 2 IF .REC_ADDR[IRC$V_NOPTRSZ]  
554      0614 2 THEN  
555      0615 2 REC_ADDR = .REC_ADDR + 1  
556      0616 2 ELSE  
557      0617 2  
558      0618 2 ! The file is a prologue 3 file, and the current array element consists  
559      0619 2 of a control byte, a one word ID, and a 2, 3, or 4 byte VBN.  
560      0620 2  
561      0621 2 IF .IFAB[IFBSB_PLG_VER] GEQU PLGSC_VER_3  
562      0622 2 THEN  
563      0623 3 REC_ADDR = .REC_ADDR + (.REC_ADDR[IRC$V_PTRSZ] + 1)
```

```

: 564 0624 2          + IRC$C_DATPTRBS3
: 565 0625 2
: 566 0626 2          | The file is a prologue 2 file, and the current array element consists
: 567 0627 2          | if a control byte, a one byte ID, and a 2, 3, or 4 byte VBN.
: 568 0628 2
: 569 0629 2          ELSE
: 570 0630 3          REC_ADDR = .REC_ADDR + (.REC_ADDR[IRC$V_PTRSZ] + 1)
: 571 0631 2          + IRC$C_DATPTRBAS;
: 572 0632 2
: 573 0633 1          END;

```

|    |    |                          |                               |               |                           |      |
|----|----|--------------------------|-------------------------------|---------------|---------------------------|------|
|    | 03 | 66                       | 04 E1 00000 RMSGETNXT_ARRAY:: |               |                           |      |
|    |    |                          | 56 D6 00004                   | BBC           | #4, (REC_ADDR), 1\$       | 0613 |
|    |    |                          | 05 00006                      | INCL          | REC_ADDR                  | 0615 |
|    |    | 03 00B7 CA 91 00007 1\$: | 0B 1F 0000C                   | RSB           |                           |      |
|    |    |                          | CMPB                          | 183(IFAB), #3 | 0621                      |      |
| 50 | 66 | 02 00                    | EF 0000E                      | BLSSU         | 2\$                       |      |
|    |    | 56 05 A046 9E 00013      | 05 00018                      | EXTZV         | #0, #2, (REC_ADDR), R0    | 0623 |
|    |    |                          | 04 A046 9E 0001E              | MOVAB         | 5(R0)[REC_ADDR], REC_ADDR | 0624 |
| 50 | 66 | 02 00                    | EF 00019 2\$:                 | RSB           |                           | 0623 |
|    |    | 56 04 A046 9E 00023      | 05 00023                      | EXTZV         | #0, #2, (REC_ADDR), R0    | 0630 |
|    |    |                          |                               | MOVAB         | 4(R0)[REC_ADDR], REC_ADDR | 0631 |
|    |    |                          |                               | RSB           |                           | 0633 |

; Routine Size: 36 bytes,    Routine Base: RMSRMS3 + 00DE

```

575 0634 1 %SBTTL 'RMSREC_OVHD'
576 0635 1 GLOBAL ROUTINE RMSREC_OVHD (REC_TYPE; REC_SIZE) : RL$REC_OVHD =
577 0636 1
578 0637 1 ++
579 0638 1
580 0639 1 FUNCTIONAL DESCRIPTION:
581 0640 1
582 0641 1 This routine accepts as input the address of a record and the record's
583 0642 1 type, and returns the number of bytes of record overhead after setting
584 0643 1 the global register REC_SIZE to the size of record itself minus this
585 0644 1 overhead.
586 0645 1
587 0646 1 CALLING SEQUENCE:
588 0647 1
589 0648 1 RMSREC_OVHD()
590 0649 1
591 0650 1 INPUT PARAMETERS:
592 0651 1 REC_TYPE
593 0652 1 < 0 - type of record REC_ADDR points to
594 0653 1 = 0 - SIDR
595 0654 1 > 0 - primary data record
596 0655 1 - index record
597 0656 1
598 0657 1 IMPLICIT INPUTS:
599 0658 1
600 0659 1 IDX_DFN - address of index descriptor for this key
601 0660 1 IDX$B_DATBKTYp - type of data bucket
602 0661 1 IDX$V_KEY_COMPR - if set, index key compression is enabled
603 0662 1 IDX$B_KEY$Z - key size
604 0663 1
605 0664 1 IFAB - address of IFAB
606 0665 1 IFBSW_LRL - record size if fixed length record format
607 0666 1 IFBSB_RFMOrg - record format of primary data records
608 0667 1 IFBSB_PLG_Ver - prologue version of file
609 0668 1
610 0669 1 REC_ADDR - address of record
611 0670 1
612 0671 1 OUTPUT PARAMETERS:
613 0672 1 NONE
614 0673 1
615 0674 1 IMPLICIT OUTPUTS:
616 0675 1 REC_SIZE - size of the entire record minus the overhead
617 0676 1
618 0677 1 ROUTINE VALUE:
619 0678 1
620 0679 1 The number of bytes of record overhead. This overhead never includes
621 0680 1 the key or key compression overhead.
622 0681 1
623 0682 1 SIDE EFFECTS:
624 0683 1 NONE
625 0684 1
626 0685 1
627 0686 1
628 0687 2 BEGIN
629 0688 2
630 0689 2 EXTERNAL REGISTER
631 0690 2 R_IDX_DFN_STR,

```

```
632      0691 2      R_IFAB_STR,  
633      0692 2      R_REC_ADDR_STR;  
634      0693 2  
635      0694 2      LOCAL  
636      0695 2      OVERHEAD:  
637      0696 2  
638      0697 2      ! CASE 1: Determine size and overhead for an INDEX RECORD.  
639      0698 2  
640      0699 2      IF .REC_TYPE GTR 0  
641      0700 2      THEN  
642      0701 3      BEGIN ! CASE 1  
643      0702 3  
644      0703 3  
645      0704 3  
646      0705 3  
647      0706 3  
648      0707 3  
649      0708 3  
650      0709 3  
651      0710 3  
652      0711 3      IF .IFAB[IFB$B_PLG_VER] GEQ PLGSC_VER_3  
653      0712 4      THEN  
654      0713 4      BEGIN  
655      0714 4      OVERHEAD = 0;  
656      0715 4      IF .IDX_DFN[IDX$V_IDX_COMPR]  
657      0716 4      THEN  
658      0717 4      REC_SIZE = .(.REC_ADDR)<0,8> + 2  
659      0718 4      ELSE  
660      0719 4      REC_SIZE = .IDX_DFN[IDX$B_KEYSZ];  
661      0720 4      END  
662      0721 4  
663      0722 4  
664      0723 4  
665      0724 4  
666      0725 3  
667      0726 4  
668      0727 4  
669      0728 4  
670      0729 4  
671      0730 3  
672      0731 3  
673      0732 3  
674      0733 3  
675      0734 2  
676      0735 2      ELSE IF .REC_TYPE EQL 0  
677      0736 2      THEN  
678      0737 3      BEGIN ! CASE 2  
679      0738 3  
680      0739 3  
681      0740 3  
682      0741 3  
683      0742 3  
684      0743 4  
685      0744 4  
686      0745 4  
687      0746 4  
688      0747 4      ! Determine the amount of record overhead in a prologue 3 primary  
               data record.  
               ! CASE 2: Return size and overhead for a PRIMARY DATA RECORD.  
               ! If the record is not a RRV then the amount of record overhead  
               ! will depend on whether the file contains fixed length data  
               ! records with neither the primary key nor the data portion
```

```
: 689      0748 4 compressed, or any other type of record. The difference is
: 690      0749 4 in whether a two-byte record size overhead field is present
: 691      0750 4 or not.
: 692      0751 4
: 693      0752 4 If the file contains fixed length records and both key and
: 694      0753 4 data compression are disabled, then the record's size (minus
: 695      0754 4 the record overhead's contribution) is a known constant;
: 696      0755 4 otherwise, for the remaining record types the size maybe
: 697      0756 4 obtained from the last two bytes of the record's overhead
: 698      0757 4
: 699      0758 4 IF NOT .REC_ADDR[IRC$V_RRV]
: 700      0759 4 THEN
: 701      0760 6     IF NOT ((.IFAB[IFBSB_RFMOORG] EQLU FABSC FIX)
: 702      0761 5         AND (.IDX_DFN[IDXSB_DATBKTYP] EQLU IDXSC_NCMPNCMP))
: 703      0762 4 THEN
: 704      0763 5     BEGIN
: 705      0764 5     OVERHEAD = IRC$C_VAROVHSZ3;
: 706      0765 6     REC_SIZE = .(.REC_ADDR + .OVERHEAD
: 707      0766 5             - IRC$C_DATSZFLD)<0,16>;
: 708      0767 5     END
: 709      0768 4 ELSE
: 710      0769 5     BEGIN
: 711      0770 5     OVERHEAD = IRC$C_FIXOVHSZ3;
: 712      0771 5     IF NOT .REC_ADDR[IRC$V_DELETED]
: 713      0772 5     THEN
: 714      0773 5     REC_SIZE = .IFAB[IFBSW_LRL]
: 715      0774 5     ELSE
: 716      0775 5     REC_SIZE = .IDX_DFN[IDXSB_KEYSZ];
: 717      0776 5     END
: 718      0777 5
: 719      0778 5 ! If the record is a RRV then the overhead will consist of a
: 720      0779 5 one-byte control byte, a two-byte record ID, and a six-byte
: 721      0780 5 RRV provided the record contains an RRV. Note that for an
: 722      0781 5 RRV, the record's size is always 0.
: 723      0782 5
: 724      0783 4 ELSE
: 725      0784 4     IF .REC_ADDR[IRC$V_NOPTRSZ]
: 726      0785 4     THEN
: 727      0786 4     OVERHEAD = IRC$C_DATOVHSZ3
: 728      0787 4     ELSE
: 729      0788 4     OVERHEAD = IRC$C_RRVOVHSZ3;
: 730      0789 4
: 731      0790 4 END
: 732      0791 4
: 733      0792 4 ! Determine the amount of record overhead in a prologue 1 or 2
: 734      0793 4 primary data record.
: 735      0794 4
: 736      0795 3 ELSE
: 737      0796 4     BEGIN
: 738      0797 4
: 739      0798 4     If the record is not a RRV then the amount of record overhead
: 740      0799 4 will depend on whether the file contains fixed length data
: 741      0800 4 records or variable length data records. The difference is
: 742      0801 4 in whether a two-byte record size overhead field is present
: 743      0802 4 or not.
: 744      0803 4
: 745      0804 4     If the file contains fixed length records then the record's
```

```
; 746      0805 4      ! size (minus the record overhead's contribution) is a known
747      0806 4      constant; otherwise, for variable length records the size
748      0807 4      maybe obtained from the last two bytes of the record's
749      0808 4      overhead.
750      0809 4
751      0810 4      IF NOT .REC_ADDR[IRC$V_RRV]
752      0811 4      THEN
753      0812 4          IF .IFAB[IFBSB_RFMRG] EQLU FABSC_FIX
754      0813 4          THEN
755      0814 5              BEGIN
756      0815 5                  OVERHEAD = IRC$C_FIXOVHDSZ;
757      0816 5                  REC_SIZE = .IFAB[IFBSW_LRL];
758      0817 5              END
759      0818 4          ELSE
760      0819 5              BEGIN
761      0820 5                  OVERHEAD = IRC$C_VAROVHDSZ;
762      0821 6                  REC_SIZE = .(.REC_ADDR + .OVERHEAD
763      0822 5                      - IRC$C_DATSZFLD)<0,16>;
764      0823 5              END
765      0824 5
766      0825 5          ! If the record is a RRV then the overhead will consist of a
767      0826 5          one-byte control byte, a one-byte record ID, and a five-byte
768      0827 5          RRV provided the record contains an RRV. The record's size
769      0828 5          is 0.
770      0829 5
771      0830 4          ELSE
772      0831 4              IF .REC_ADDR[IRC$V_NOPTRSZ]
773      0832 4              THEN
774      0833 4                  OVERHEAD = IRC$C_DATOVHDSZ
775      0834 4              ELSE
776      0835 4                  OVERHEAD = IRC$C_RRVOVHDSZ;
777      0836 4
778      0837 3          END;
779      0838 3      END ! CASE 2
780      0839 3
781      0840 3      ! CASE 3: Determine size and overhead for a SIDR.
782      0841 3
783      0842 2      ELSE ! .REC_TYPE LSS 0
784      0843 3          BEGIN ! CASE 3
785      0844 3
786      0845 3          ! The overhead of a prologue 3 SIDR is fixed regardless of whether
787      0846 3          duplicate alternate keys are, or are not allowed.
788      0847 3
789      0848 3          IF .IFAB[IFBSB_PLG_VER] GEQ PLG$C_VER_3
790      0849 3          THEN
791      0850 3              OVERHEAD = IRC$C_SDROVHSZ3
792      0851 3
793      0852 3          ! The overhead of a prologue 2 SIDR will depend upon whether
794      0853 3          duplicate alternate keys are allowed or not, and if so, whether
795      0854 3          an unused duplicate count field is present in the record or not.
796      0855 3
797      0856 3
798      0857 3          ELSE
799      0858 3              IF .REC_ADDR[IRC$V_NODUPCNT]
800      0859 3              THEN
801      0860 3                  OVERHEAD = IRC$C_DATOVHDSZ + IRC$C_DATSZFLD
802      0861 3              ELSE
803      0862 3                  OVERHEAD = IRC$C_DATOVHDSZ + IRC$C_DATSZFLD
```

```

803    0862 3      + IRC$C_DCNTSZFLD;
804    0863 3
805    0864 3      ! The record's size (minus the overhead's contribution) may be
806    0865 3      obtained from the last two bytes of the SIDL overhead regardless
807    0866 3      of the prologue version of the file.
808    0867 3
809    0868 3      REC_SIZE = .(.REC_ADDR + .OVERHEAD - 2)<0,16>;
810    0869 2      END; ! CASE 3
811    0870 2
812    0871 2
813    0872 2      ! Return the number of bytes of record overhead.
814    0873 2
815    0874 2      RETURN .OVERHEAD;
816    0875 1      END;

```

|  |  |    |       | 52 DD 00000 RMSREC_OVHD:: |                                    |  |      |
|--|--|----|-------|---------------------------|------------------------------------|--|------|
|  |  |    | 52    | 00B7 CA 9E 00002          | PUSHL R2                           |  | 0635 |
|  |  |    |       | 51 D5 00007               | MOVAB 183(IFAB), R2                |  | 0710 |
|  |  |    |       | 1E 15 00009               | TSTL REC_TYPE                      |  | 0699 |
|  |  |    | 03    | 62 91 0000B               | BLEQ 2\$                           |  | 0710 |
|  |  |    |       | 0F 1F 0000E               | CMPB (R2), #3                      |  | 0713 |
|  |  |    |       | 50 D4 00010               | BLSSU 1\$                          |  | 0714 |
|  |  | 35 | 1C A7 | 03 E1 00012               | CLRL OVERHEAD                      |  | 0716 |
|  |  |    |       | 51 66 9A 00017            | BBC #3, 28(IDX_DFN), 5\$           |  | 0727 |
|  |  |    |       | 51 02 C0 0001A            | MOVZBL (REC_ADDR), REC_SIZE        |  | 0728 |
|  |  |    |       | 65 11 0001D               | ADDL2 #2, REC_SIZE                 |  | 0734 |
|  |  | 50 | 66 02 | 00 EF 0001F               | BRB 13\$                           |  | 0741 |
|  |  |    |       | 50 03 C0 00024            | EXTZV #0, #2, (REC_ADDR), OVERHEAD |  | 0758 |
|  |  |    |       | 23 11 00027               | ADDL2 #3, OVERHEAD                 |  | 0760 |
|  |  |    |       | 5B 12 00029               | BRB 5\$                            |  | 0761 |
|  |  |    | 03    | 62 91 0002B               | BNEQ 14\$                          |  | 0764 |
|  |  |    |       | 30 1F 0002E               | CMPB (R2), #3                      |  | 0771 |
|  |  | 1E | 66 01 | 03 E0 00030               | BLSSU 8\$                          |  | 0775 |
|  |  |    |       | 01 50 AA 91 00034         | BBS #3, (REC_ADDR), 6\$            |  | 0784 |
|  |  |    |       | 06 12 00038               | CMPB 80(IFAB), #1                  |  | 0786 |
|  |  |    | 06 29 | A7 91 0003A               | BNEQ 3\$                           |  | 0791 |
|  |  |    |       | 05 13 0003E               | CMPB 41(IDX_DFN), #6               |  | 0810 |
|  |  |    |       | 50 0B D0 00040            | BEQL 4\$                           |  | 0812 |
|  |  |    |       | 50 57 11 00043            | MOVL #11, OVERHEAD                 |  | 0815 |
|  |  | 21 | 66 51 | 09 D0 00045               | BRB 17\$                           |  |      |
|  |  |    |       | 02 E1 00048               | MOVL #9, OVERHEAD                  |  |      |
|  |  |    |       | 51 9A 0004C               | BBC #2, (REC_ADDR), 9\$            |  |      |
|  |  |    |       | 51 11 00050               | MOVZBL 32(IDX_DFN), REC_SIZE       |  |      |
|  |  | 05 | 66 50 | 04 E1 00052               | BRB 18\$                           |  |      |
|  |  |    |       | 03 D0 00056               | MOVL #4, (REC_ADDR), 7\$           |  |      |
|  |  |    |       | 48 11 00059               | BBC #3, OVERHEAD                   |  |      |
|  |  |    | 50    | 09 D0 00058               | MOVL #9, OVERHEAD                  |  |      |
|  |  |    |       | 43 11 0005E               | BRB 18\$                           |  |      |
|  |  | 14 | 66 01 | 03 E0 00060               | MOVL #3, (REC_ADDR), 11\$          |  |      |
|  |  |    |       | AA 91 00064               | BBS 80(IFAB), #1                   |  |      |
|  |  |    |       | 09 12 00068               | BNEQ 10\$                          |  |      |
|  |  |    | 50    | 07 D0 0006A               | MOVL #7, OVERHEAD                  |  |      |

|    |    |         |                   |                               |        |
|----|----|---------|-------------------|-------------------------------|--------|
|    | 51 | 52      | AA 3C 0006D 9\$:  | MOVZWL 82(IFAB), REC_SIZE     | : 0816 |
|    |    |         | 30 11 00071       | BRB 18\$                      | : 0812 |
|    | 50 |         | 09 D0 00073 10\$: | MOVL #9 OVERHEAD              | : 0820 |
|    |    |         | 24 11 00076       | BRB 17\$                      | : 0821 |
| 05 | 66 |         | 04 E1 00078 11\$: | BBC #4, (REC_ADDR), 12\$      | : 0831 |
|    | 50 |         | 02 D0 0007C       | MOVL #2 OVERHEAD              | : 0833 |
|    |    |         | 22 11 0007F       | BRB 18\$                      | :      |
|    | 50 |         | 07 D0 00081 12\$: | MOVL #7 OVERHEAD              | : 0835 |
|    |    |         | 1D 11 00084 13\$: | BRB 18\$                      | : 0734 |
|    | 03 |         | 62 91 00086 14\$: | CMPB (R2), #3                 | : 0848 |
|    |    |         | 05 1F 00089       | BLSSU 15\$                    | :      |
|    | 50 |         | 02 D0 0008B       | MOVL #2 OVERHEAD              | : 0850 |
|    |    |         | 0C 11 0008E       | BRB 17\$                      | :      |
| 05 | 66 |         | 04 E1 00090 15\$: | BBC #4, (REC_ADDR), 16\$      | : 0857 |
|    | 50 |         | 04 D0 00094       | MOVL #4 OVERHEAD              | : 0859 |
|    |    |         | 03 11 00097       | BRB 17\$                      | :      |
|    | 50 |         | 08 D0 00099 16\$: | MOVL #8, OVERHEAD             | : 0862 |
|    |    | FE A046 | 9F 0009C 17\$:    | PUSHAB -2(OVERHEAD)[REC_ADDR] | : 0868 |
|    | 51 |         | 9E 3C 000A0       | MOVZWL a(SP)+, REC_SIZE       | :      |
|    |    |         | 04 BA 000A3 18\$: | POPR #^M<R2>                  | : 0875 |
|    |    |         | 05 000A5          | RSB                           | :      |

: Routine Size: 166 bytes.    Routine Base: RMSRMS3 + 0102

```
818 0876 1 ZSBTTL 'RMSSIDR END'
819 0877 1 GLOBAL ROUTINE RMSSIDR_END : RL$RABREG_67 =
820 0878 1 ++
821 0879 1
822 0880 1
823 0881 1 FUNCTIONAL DESCRIPTION:
824 0882 1
825 0883 1 The purpose of this routine is to return the address of the first
826 0884 1 past the end of the current SIDR.
827 0885 1
828 0886 1 CALLING SEQUENCE:
829 0887 1 BSBW RMSSIDR_END()
830 0888 1
831 0889 1 INPUT PARAMETERS:
832 0890 1 NONE
833 0891 1
834 0892 1 IMPLICIT INPUTS:
835 0893 1
836 0894 1 REC_ADDR - address of the SIDR
837 0895 1
838 0896 1 OUTPUT PARAMETERS:
839 0897 1 NONE
840 0898 1
841 0899 1 IMPLICIT OUTPUTS:
842 0900 1 NONE
843 0901 1
844 0902 1 ROUTINE VALUE:
845 0903 1
846 0904 1 Address of the first byte past the current SIDR's last array element.
847 0905 1
848 0906 1 SIDE EFFECTS:
849 0907 1 NONE
850 0908 1
851 0909 1 --
852 0910 1
853 0911 2 BEGIN
854 0912 2
855 0913 2 EXTERNAL REGISTER
856 0914 2 COMMON RABREG,
857 0915 2 R_IDX_DFN_STR,
858 0916 2 R_REC_ADDR_STR;
859 0917 2
860 0918 2 LOCAL
861 0919 2 END_OF_SIDR,
862 0920 2 SAVE_REC_ADDR;
863 0921 2
864 0922 2 ! Save the address of the beginning of the SIDR.
865 0923 2
866 0924 2 SAVE_REC_ADDR = .REC_ADDR;
867 0925 2
868 0926 2 ! Obtain the address of the next record in the bucket.
869 0927 2
870 0928 2 RMSGETNEXT_REC();
871 0929 2 END_OF_SIDR = .REC_ADDR;
872 0930 2
873 0931 2 ! Restore the address of the beginning of the SIDR, and return the address
874 0932 2 ! of the next record - effectively the end of the current SIDR.
```

: 875 0933 2 !  
: 876 0934 2 REC\_ADDR = .SAVE\_REC\_ADDR;  
: 877 0935 2 RETURN .END\_OF\_SIDR;  
: 878 0936 2  
: 879 0937 1 END:

52 DD 00000 RM\$SIDR\_END::  
52 56 D0 00002 PUSHL R2  
FEFF 30 00005 MOVL REC\_ADDR, SAVE\_REC\_ADDR : 0877  
50 56 D0 00008 BSBW RMS\$GETNEXT REC : 0924  
56 52 D0 0000B MOVL REC\_ADDR, END\_OF\_SIDR : 0928  
04 BA 0000E POPR SAVE\_REC\_ADDR, REC\_ADDR : 0929  
05 00010 RSB #^M<R2> : 0934  
: 0937

: Routine Size: 17 bytes, Routine Base: RMS\$RMS3 + 01A8

881 0938 1 %SBTTL 'RM\$SIDR FIRST'  
882 0939 1 GLOBAL ROUTINE RMSSIDR\_FIRST (FLAGS;RFA\_VBN, RFA\_ID) : RL\$SIDR\_FIRST =  
883 0940 1 !++  
884 0941 1 FUNCTIONAL DESCRIPTION:  
885 0942 1  
886 0943 1 The purpose of this routine is to return the address and optionally the  
887 0944 1 RFA pointer of the first element of the inputed SIDR. The components of  
888 0945 1 the RFA pointer are returned separately (VBN and ID), and in the RMS  
889 0946 1 cluster environment, uniquely identifies the SIDR from among all SIDRs  
890 0947 1 with that key value.  
891 0948 1  
892 0949 1  
893 0950 1  
894 0951 1 CALLING SEQUENCE:  
895 0952 1 BSBW RMSSIDR\_FIRST()  
896 0953 1  
897 0954 1 INPUT PARAMETERS:  
898 0955 1  
899 0956 1 FLAGS - if 1, return the RFA pointer of the first element  
900 0957 1  
901 0958 1 IMPLICIT INPUTS:  
902 0959 1  
903 0960 1 IDX\_DFN - index descriptor for the SIDR  
904 0961 1 IDX\$V\_KEY COMPR - if set, SIDR key compression is enabled  
905 0962 1 IDX\$B\_KEYSZ - size of a fully expanded SIDR key  
906 0963 1  
907 0964 1 REC\_ADDR - address of the SIDR  
908 0965 1  
909 0966 1 OUTPUT PARAMETERS:  
910 0967 1  
911 0968 1 RFA\_VBN - VBN of the SIDR's first array element RFA pointer  
912 0969 1 RFA\_ID - ID of the SIDR's first array element RFA pointer  
913 0970 1  
914 0971 1 IMPLICIT OUTPUTS:  
915 0972 1 NONE  
916 0973 1  
917 0974 1 ROUTINE VALUE:  
918 0975 1 Address of the first array element of the SIDR.  
919 0976 1  
920 0977 1  
921 0978 1 SIDE EFFECTS:  
922 0979 1  
923 0980 1 AP is trashed  
924 0981 1  
925 0982 1 --  
926 0983 1  
927 0984 2 BEGIN  
928 0985 2  
929 0986 2 BUILTIN  
930 0987 2 AP;  
931 0988 2  
932 0989 2 EXTERNAL REGISTER  
933 0990 2 COMMON RABREG,  
934 0991 2 R\_IDX\_DFN\_STR,  
935 0992 2 R\_REC\_ADDR\_STR;  
936 0993 2  
937 0994 2 LOCAL

```

938 0995 2 BEGIN_SIDR,
939 0996 2 FIRST_ELEMENT;
940 0997 2
941 0998 2 ! After saving the address of the SIDR, position to the SIDR's first
942 0999 2 array element.
943 1000 2
944 1001 2 BEGIN_SIDR = .REC_ADDR;
945 1002 2
946 1003 2 REC_ADDR = .REC_ADDR + RMSREC_OVHD(-1);
947 1004 2
948 1005 2 IF .IDX_DFN[IDX$V_KEY_COMPRESS]
949 1006 2 THEN REC_ADDR = .REC_ADDR + (.REC_ADDR)<0,8> + IRC$C_KEYCMPOVH
950 1007 2 ELSE REC_ADDR = .REC_ADDR + .IDX_DFN[IDX$B_KEYSZ];
951 1008 2
952 1009 2
953 1010 2
954 1011 2 ! If the caller has requested that the VBN and ID of the first element's
955 1012 2 RFA pointer be returned, extract them from the RFA pointer of the first
956 1013 2 element of the SIDR.
957 1014 2
958 1015 2 IF .FLAGS<0,1>
959 1016 2 THEN RM$EXT_ARRY_RFA (RFA_VBN, RFA_ID);
960 1017 2
961 1018 2
962 1019 2 ! Restore to REC_ADDR the address of the SIDR and return the address of the
963 1020 2 first element.
964 1021 2
965 1022 2 FIRST_ELEMENT = .REC_ADDR;
966 1023 2 REC_ADDR = .BEGIN_SIDR;
967 1024 2
968 1025 2 RETURN .FIRST_ELEMENT;
969 1026 2
970 1027 1 END;

```

|    |    | RMSSIDR_FIRST:: |                  |      |                                 |        |
|----|----|-----------------|------------------|------|---------------------------------|--------|
|    |    | 5E              | 08 C2 00000      |      | SUBL2 #8, SP                    | : 0939 |
|    |    | 52              | 56 D0 00003      |      | MOVL REC_ADDR, BEGIN_SIDR       | : 1001 |
|    |    | 51              | 01 CE 00006      |      | MNEGL #1, R1                    | : 1003 |
|    |    |                 | FF3D 30 00009    |      | BSBW RM\$REC_OVHD               |        |
| 0A | 1C | 56              | 50 C0 0000C      |      | ADDL2 R0, REC_ADDR              |        |
|    |    | A7              | 06 E1 0000F      |      | BBC #6, 28(IDX_DFN), 1\$        | : 1005 |
|    |    | 50              | 66 9A 00014      |      | MOVZBL (REC_ADDR), R0           | : 1007 |
|    |    | 56              | 02 A046 9E 00017 |      | MOVAB 2(R0)[REC_ADDR], REC_ADDR |        |
|    |    |                 | 07 11 0001C      |      | BRB 2\$                         |        |
|    |    | 50              | 20 A7 9A 0001E   | 1\$: | MOVZBL 32(IDX_DFN), R0          | : 1009 |
|    |    | 56              | 50 C0 00022      |      | ADDL2 R0, REC_ADDR              |        |
|    |    | 0B              | 0C AE E9 00025   | 2\$: | BLBC FLÁGS, 3\$                 | : 1015 |
|    |    |                 | 5E DD 00029      |      | PUSHL SP                        | : 1017 |
|    |    | 08              | AE 9F 0002B      |      | PUSHAB RFA_VBN                  |        |
|    |    |                 | FE8D 30 0002E    |      | BSBW RM\$EXT_ARRY_RFA           |        |
|    |    | 5E              | 08 C0 00031      |      | ADDL2 #8, SP                    |        |
|    |    | 50              | 56 D0 00034      | 3\$: | MOVL REC_ADDR, FIRST_ELEMENT    | : 1022 |
|    |    | 56              | 52 D0 00037      |      | MOVL BEGIN_SIDR, REC_ADDR       | : 1023 |

RM3NEXTRE  
V04-000

RM\$SIDR\_FIRST

B 9  
16-Sep-1984 01:53:40 VAX-11 Bliss-32 V4.0-742  
14-Sep-1984 13:01:30 [RMS.SRC]RM3NEXTRE.B32;1

Page 26  
(8)

RM3C  
V04-

52            BE DO 0003A            MOVL RFA\_ID, R2  
51            BE DO 0003D            MOVL RFA\_VBN, R1  
              05 00040            RSB

; 1027  
;  
;

; Routine Size: 65 bytes,    Routine Base: RM\$RMS3 + 01B9

; 971        1028 1  
; 972        1029 1  
; 973        1030 1 END  
; 974        1031 1  
; 975        1032 0 ELUDOM

#### PSECT SUMMARY

| Name     | Bytes | Attributes   |
|----------|-------|--|
| RM\$RMS3 | 506   | NOVEC,NOWRT, RD , EXE,NOSHR, GBL, REL, CON, PIC,ALIGN(2) |

#### Library Statistics

| File                             | ----- Symbols -----            | Pages Mapped | Processing Time |
|----------------------------------|--------------------------------|--------------|-----------------|
|                                  | Total      Loaded      Percent |              |                 |
| \$_255\$DUA28:[RMS.OBJ]RMS.L32;1 | 3109      64      2            | 154          | 00:00.4         |

; Re  
;  
;

#### COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LISS:RM3NEXTRE/OBJ=OBJ\$:RM3NEXTRE MSRC\$:RM3NEXTRE/UPDATE=(ENH\$:RM3NEXTRE)

Size:        506 code + 0 data bytes  
Run Time:    00:14.8  
Elapsed Time: 00:28.1  
Lines/CPU Min: 4180  
Lexemes/CPU-Min: 13296  
Memory Used: 100 pages  
Compilation Complete

0326 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY

RM3NEXTRE  
LIS

RM3OPEN  
LIS

RM3POSRFA  
LIS

RM3PCKUP  
LIS

RM3POSKEY  
LIS

RM3POSSEQ  
LIS