


```

1 0001 0 MODULE RM3POSSEQ (LANGUAGE (BLISS32) ,
2 0002 0 IDENT = 'V04-000'
3 0003 0 ) =
4 0004 1 BEGIN
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY *
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS. *
10 0010 1 * ALL RIGHTS RESERVED. *
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED *
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE *
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER *
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY *
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY *
17 0017 1 * TRANSFERRED. *
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE *
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT *
21 0021 1 * CORPORATION. *
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS *
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL. *
25 0025 1 *
26 0026 1 *
27 0027 1 *****
28 0028 1
29 0029 1 ++
30 0030 1
31 0031 1 FACILITY: RMS32 Index Sequential File Organization
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1 Routines to position sequentially to the next record
35 0035 1
36 0036 1
37 0037 1 ENVIRONMENT:
38 0038 1
39 0039 1 VAX/VMS Operating System
40 0040 1
41 0041 1 --
42 0042 1
43 0043 1
44 0044 1 AUTHOR: Todd M. Katz CREATION DATE: 23-Jun-1982
45 0045 1
46 0046 1
47 0047 1 MODIFIED BY:
48 0048 1
49 0049 1 V03-016 TSK0001 Tamar Krichevsky 15-Jun-1983
50 0050 1 Change addressing mode of RMSRU_RECLAIM to long relative.
51 0051 1
52 0052 1 V03-015 MCN0002 Maria del C. Nasr 24-Mar-1983
53 0053 1 More linkages reorganization.
54 0054 1
55 0055 1 V03-014 TMK0009 Todd M. Katz 11-Mar-1983
56 0056 1 Make sure that all bucket locks are taken out as exclusive
57 0057 1 whenever the possibility exists that some reclamation maybe

```

: R

```

58 0058 1 done. Reclamation is possible whenever the positioning is by
59 0059 1 alternate key of reference, or whenever the file has been
60 0060 1 write accessed and is Recovery Unit Journallable.
61 0061 1
62 0062 1 V03-013 MCN0001 Maria del C. Nasr 28-Feb-1983
63 0063 1 Reorganize linkages
64 0064 1
65 0065 1 V03-012 TMK0008 Todd M. Katz 24-Feb-1983
66 0066 1 Fix a bug in and add several performance enhancements to
67 0067 1 RMSFIND_NONDEL. The bug was that when this routine exhausts the
68 0068 1 current bucket in its search for a non-deleted primary data
69 0069 1 record and continues with the next bucket it should be resetting
70 0070 1 IRBSL_LST_NCMP to point to the first record in the next bucket
71 0071 1 when primary key compression is enabled. Because it was not,
72 0072 1 when the time came to re-expand the key of the non-deleted
73 0073 1 primary data record positioned to, this key expansion was done
74 0074 1 incorrectly. The enhancements consist of making sure as RMS
75 0075 1 scans the current bucket for a non-deleted primary data record
76 0076 1
77 0077 1
78 0078 1 V03-011 TMK0007 Todd M. Katz 17-Jan-1983
79 0079 1 Add support for Recovery Unit Journalling and RU ROLLBACK
80 0080 1 Recovery of ISAM files. Support involves modifications to
81 0081 1 RMSPOS_SEQ and RMSFIND_NONDEL.
82 0082 1
83 0083 1 The purpose of the routines within this module is to find the
84 0084 1 next non-deleted primary data record sequentially. If during
85 0085 1 its search RMS encounters records that are marked RU DELETE, it
86 0086 1 will try and delete them for good at this time provided it has
87 0087 1 write access to the file and the Recovery Unit in which they
88 0088 1 were deleted has completed successfully.
89 0089 1
90 0090 1 if RMS is able to delete a primary data record marked RU_DELETE,
91 0091 1 then RMS proceeds to continue looking for a non-deleted primary
92 0092 1 data record just as if it had encountered a deleted record in
93 0093 1 the first place. Likewise, if RMS is unable to delete a record
94 0094 1 that is marked RU_DELETE because it does not have write access
95 0095 1 to the file, it merely continues its search with the next
96 0096 1 record. However, if RMS is unable to delete the record for good
97 0097 1 because the Recovery Unit in which it was marked RU_DELETE has
98 0098 1 not successfully terminated, then RMS returns this record as if
99 0099 1 it was the next non-deleted primary data record, and lets a
100 0100 1 higher level routine decide whether or not to wait for the
101 0101 1 Recovery Unit in which the record was deleted to complete, or to
102 0102 1 return an error to the user.
103 0103 1
104 0104 1 RMS will also re-format any records that are marked RU_UPDATE
105 0105 1 and are in a special format provided the stream has write access
106 0106 1 to the file, and the Recovery Unit in which the record was
107 0107 1 updated has terminated.
108 0108 1
109 0109 1 V03-010 TMK0006 Todd M. Katz 09-Nov-1982
110 0110 1 The routine RMSFIND_SIDR never returns an end-of-file error so
111 0111 1 remove all references to this particular error.
112 0112 1
113 0113 1 V03-009 TMK0005 Todd M. Katz 20-Oct-1982
114 0114 1 An infinite loop existed in RMSPOS_SEQ. When a search must be

```

```

115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171

```

```

made for the current primary data record by calling
RMSSEARCH_TREE and duplicates are allowed, then the duplicate
chain is searched for the current record. If the first duplicate
in the chain was not the current record, then RMS was assuming
that if it called RMSSEARCH_TREE, this routine would position
to the next record, and return status indicating whether it
too was a duplicate. This assumption is incorrect. The routine
RMSSEARCH_TREE works as follows: If the routine
RMSSEARCH_TREE is called with REC_ADDR pointing to a place in
a bucket, whose BDB's address is saved in IRB$B_CURBDB, then
RMSSEARCH_TREE determines whether or not the key of this
record matches the key in keybuffer 2 according to the search
characteristics. Thus, this routine does not position RMS to the
very next record, which was what was being incorrectly assumed, but
just checks out the key of the current record. The fix is to
explicitly position to the next primary data record if the
record currently positioned to is not the current record, before
calling RMSSEARCH_TREE to see if this record is a member of the
duplicate chain.

```

```

V03-008 TMK0004      Todd M. Katz      19-Oct-1982
I have made two changes to the routine RMSPOS_SEQ. First, if
RMS was unable to position to any record using the current NRP
information, then RMS should not be checking whether the record
it has positioned to is a non-deleted RRV (because of course,
RMS has not positioned to any record). At present, this check
is made in RMSPOS_SEQ. Second, RMS should not be releasing
buckets, and then referencing addresses within them. This also
is currently taking place within RMSPOS_SEQ.

```

```

V03-007 TMK0003      Todd M. Katz      09-Sep-1982
The field IRB$B_SRCHFLAGS is now a word in size. Fix all
references to it.

Add support for prologue 3 SIDRs. A change is required to
RMSPOS_SEQ. When positioning sequentially for the first time
following a $CONNECT or $KEWIND, RMS wants to position to
the first non-empty bucket. Because prologue 3 SIDRs do not have
control bytes, RMS can't just check for a RRV record as its
indicator of an empty bucket. It can only make this check for
primary data buckets. This change has been made. Furthermore,
RMS will also now check for an empty SIDR bucket (this was not
being done) by checking the freespace offset pointer.

```

```

V03-006 KBT0296      Keith B. Thompson  24-Aug-1982
Reorganize psects

```

```

V03-005 TMK0002      Todd M. Katz      28-Jul-1982
When positioning sequentially by alternate key, the routine
RMSFIND_SIDR is called to find the current SIDR, or to position
to the next SIDR if the current SIDR has been deleted. If a
stream's current SIDR is the last SIDR in the index, and it is
deleted by another stream, RMSFIND_SIDR returns the error
RMS$RNF because it is unable to find any SIDR in the
greater-than-or-equal search of the file it makes (using as the
search key the key of the current record) when it decides that
the stream's current SIDR has been deleted. In this case

```

```

172 0172 1 |
173 0173 1 |
174 0174 1 |
175 0175 1 |
176 0176 1 |
177 0177 1 |
178 0178 1 |
179 0179 1 |
180 0180 1 |
181 0181 1 |
182 0182 1 |
183 0247 1 |
184 0248 1 |
185 0249 1 |
186 0250 1 |
187 0251 1 |
188 0252 1 |
189 0253 1 |
190 0254 1 |
191 0255 1 |
192 0256 1 |
193 0257 1 |
194 0258 1 |
195 0259 1 |
196 0260 1 |
197 0261 1 |
198 0262 1 |
199 0263 1 |
200 0264 1 |
201 0265 1 |
202 0266 1 |
203 0267 1 |
204 0268 1 |
205 0269 1 |
206 0270 1 |
207 0271 1 |
208 0272 1 |
209 0273 1 |
210 0274 1 |
211 0275 1 |
212 0276 1 |
213 0277 1 |
214 0278 1 |
215 0279 1 |
216 0280 1 |
217 0281 1 |
218 0282 1 |
219 0283 1 |
220 0284 1 |
221 0285 1 |
222 0286 1 |
223 0287 1 |
224 0288 1 |
225 0289 1 |
226 0290 1 |
227 M 0291 1 |
228 M 0292 1 |

```

RMSPOS_SEQ should convert this error into RMSS_EOF and return
 it to indicate that the end of the file has been reached.

V03-004 TMK0001 Todd M. Katz 22-Jun-1982
 Revised entire module.

```

LIBRARY 'RMSLIB:RMS';
REQUIRE 'RMSSRC:RMSIDXDEF';
! Define default PSECTS for code
PSECT
  CODE = RMSRMS3(PSECT_ATTR),
  PLIT = RMSRMS3(PSECT_ATTR);
! Linkages
LINKAGE
  L_PRESERVE1,
  L_REC_OVHD,
  L_SIDR_FIRST,
  L_RABREG_457,
  L_RABREG_567,
  L_RABREG_67,
! Local Linkages
RL$LINKAGE = JSB ()
: GLOBAL (R_IDX_DFN, R_REC_ADDR, COMMON_RABREG);
! External Routines
EXTERNAL ROUTINE
  RMSCOMPARE_REC : RL$RABREG_67,
  RMSSEARCH_TREE : RL$RABREG_67,
  RMSFIND_BY_RFA : RL$RABREG_67,
  RMSFIND_BY_RRV : RL$RABREG_67,
  RMSGETBKT : RL$RABREG_457,
  RMSGETNEXT_REC : RL$RABREG_67,
  RMSRECORD_ID : RL$RABREG_67,
  RMSRECORD_VBN : RL$PRESERVE1,
  RMSREC_OVHD : RL$REC_OVHD,
  RMSRLSBKT : RL$PRESERVE1,
  RMSRU_RECLAIM : RL$RABREG_67 ADDRESSING_MODE( LONG_RELATIVE ),
  RMSSEARCH_SIDR : RL$RABREG_67,
  RMSSIDR_FIRST : RL$SIDR_FIRST,
  RMSSRCH_BY_KEY : RL$RABREG_567;
! Define a local macro to return on error and
! to change a 'RNF' error to 'EOF' ( this is seq. processing )
MACRO
  RET_ON_ERR_EOF (X) =
  (RETURN_ON_ERROR(X,

```

RM3POSSEQ
V04-000

F 14
16-Sep-1984 01:56:09 YAX-11 Bliss-32 V4.0-742
14-Sep-1984 13:01:34 [RMS.SRC]RM3POSSEQ.B32;1

Page 5
(1)

RM3
V04

: 229
: 230

0293 1
0294 1

IF .STATUS<0,16> EQL RMSERR(RNF) THEN STATUS = RMSERR(EOF))) %;

.....

```

: 232 0295 1 %SBTTL 'RMSCHECK_RRV'
: 233 0296 1 ROUTINE RMSCHECK_RRV : RLSLINKAGE =
: 234 0297 1 ++
: 235 0298 1
: 236 0299 1 FUNCTIONAL DESCRIPTION:
: 237 0300 1
: 238 0301 1 This routine compares the RRV pointer of the record RMS is currently
: 239 0302 1 positioned to with the RFA address of the current record, and returns
: 240 0303 1 the result of that match verifying whether or not the two are one and
: 241 0304 1 the same - ie the record RMS is currently positioned to is the current
: 242 0305 1 record.
: 243 0306 1
: 244 0307 1 CALLING SEQUENCE:
: 245 0308 1
: 246 0309 1 BSBW RMSCHECK_RRV()
: 247 0310 1
: 248 0311 1 INPUT PARAMETERS:
: 249 0312 1 NONE
: 250 0313 1
: 251 0314 1 IMPLICIT INPUTS:
: 252 0315 1
: 253 0316 1 IRAB - address of the IRAB
: 254 0317 1 IRBSW_POS_ID - ID of user data record for NRP positioning
: 255 0318 1 IRBSL_POS_VBN - VBN of user data record for NRP positioning
: 256 0319 1
: 257 0320 1 REC_ADDR - address of primary data record
: 258 0321 1
: 259 0322 1 OUTPUT PARAMETERS:
: 260 0323 1 NONE
: 261 0324 1
: 262 0325 1 IMPLICIT OUTPUTS:
: 263 0326 1 NONE
: 264 0327 1
: 265 0328 1 ROUTINE VALUE:
: 266 0329 1
: 267 0330 1 SUC - the RRV of the current primary data record matches the current
: 268 0331 1 records RFA address.
: 269 0332 1
: 270 0333 1 0 - the RRV of the current primary data record does not match the
: 271 0334 1 current records RFA address.
: 272 0335 1
: 273 0336 1 SIDE EFFECTS:
: 274 0337 1
: 275 0338 1 AP is trashed.
: 276 0339 1
: 277 0340 1 --
: 278 0341 1
: 279 0342 2 BEGIN
: 280 0343 2
: 281 0344 2 BUILTIN
: 282 0345 2 AP;
: 283 0346 2
: 284 0347 2 EXTERNAL REGISTER
: 285 0348 2 COMMON RAB STR,
: 286 0349 2 R_IDX_DFN STR,
: 287 0350 2 R_REC_ADDR;
: 288 0351 2

```

```

: 289 0352 2 GLOBAL REGISTER
: 290 0353 2 R_BDB;
: 291 0354 2
: 292 0355 2 AP = 3;
: 293 0356 2
: 294 0357 2 ! If the two addresses match, return success. The record RMS is positioned
: 295 0358 2 ! to is in fact the current record.
: 296 0359 2
: 297 0360 2 IF (.IRAB[IRBSW_POS_ID] EQL RMSRECORD_ID())
: 298 0361 2 AND
: 299 0362 2 (.IRAB[IRBSL_POS_VBN] EQL RMSRECORD_VBN())
: 300 0363 2 THEN
: 301 0364 2 RETURN RMSSUC()
: 302 0365 2
: 303 0366 2 ! Otherwise, the record RMS is currently positioned to is not the current
: 304 0367 2 ! record
: 305 0368 2
: 306 0369 2 ELSE
: 307 0370 2 RETURN 0;
: 308 0371 2
: 309 0372 1 END;

```

! { of routine }

```

.TITLE RM3POSSEQ
.IDENT \V04-000\

.EXTRN RMSCOMPARE_REC, RMSSEARCH_TREE
.EXTRN RMSFIND_BY_RFA, RMSFIND_BY_RRV
.EXTRN RMSGETBKT, RMSGETNEXT_REC
.EXTRN RMSRECORD_ID, RMSRECORD_VBN
.EXTRN RMSREC_OVRD, RMSRLSBKT
.EXTRN RMSRU_RECLAIM, RMSSEARCH_SIDR
.EXTRN RMSSIDR_FIRST, RMSSRCH_BY_KEY

.PSECT RMSRMS3,NGWRT, GBL, PIC,2

```

			54	DD	0000	RMSCHECK_RRV:			
						PUSHL	R4	:	0296
	5C		03	D0	00002	MOVL	#3, AP	:	0355
			0000G	30	00005	BSBW	RMSRECORD_ID	:	0360
50	00BA	C9	10	00	ED 00008	CMPZV	#0, #16, T86(IRAB), R0	:	
				0F	12 0000F	BNEQ	1\$:	
			0000G	30	00011	BSBW	RMSRECORD_VBN	:	0362
	50	00AC	C9	D1	00014	CMPL	172(IRAB), R0	:	
				05	12 00019	BNEQ	1\$:	
	50		01	D0	0001B	MOVL	#1, R0	:	0370
				02	11 0001E	BRB	2\$:	
				50	D4 00020	CLRL	R0	:	
				10	BA 00022	POPR	#*M<R4>	:	0372
				05	00024	RSB		:	

; Routine Size: 37 bytes, Routine Base: RMSRMS3 + 0000

```

311 0373 1 %SBTTL 'RMS$IND_NONDEL'
312 0374 1 ROUTINE RMS$IND_NONDEL (FLAG) : RL$LINKAGE =
313 0375 1 +-
314 0376 1
315 0377 1 FUNCTIONAL DESCRIPTION:
316 0378 1
317 0379 1 This routine searches for the next next non-deleted primary data record
318 0380 1 in the file. If the input parameter is 1, the search starts with the
319 0381 1 record RMS is currently positioned to, but if its 0, the search will
320 0382 1 start with the very next bucket.
321 0383 1
322 0384 1 If RMS encounters a record that is marked RU_DELFTE and the Recovery
323 0385 1 Unit in which the record was deleted it still active, then RMS returns
324 0386 1 this record as the next non-deleted primary data record and lets a
325 0387 1 higher level routine decide what to do. If the Recovery Unit in which
326 0388 1 the record was deleted has successfully terminated, then RMS will
327 0389 1 continue its search for a non-deleted primary data record after deleting
328 0390 1 this RU_DELETED record (the latter if it has write access to the file).
329 0391 1
330 0392 1 If RMS encounters a record that is marked RU_UPDATE and is in a special
331 0393 1 format then RMS will return this record as the next non-deleted primary
332 0394 1 data record after reformatting it. The reformatting is done if RMS has
333 0395 1 write access to the file, and the Recovery Unit in which it was updated
334 0396 1 has successfully terminated.
335 0397 1
336 0398 1 CALLING SEQUENCE:
337 0399 1
338 0400 1 BSBW RMS$IND_NONDEL()
339 0401 1
340 0402 1 INPUT PARAMETERS:
341 0403 1
342 0404 1 FLAG - if 1, start search with the record RMS is currently
343 0405 1 positioned to
344 0406 1
345 0407 1 IMPLICIT INPUTS:
346 0408 1
347 0409 1 IDX_DFN - address of index descriptor
348 0410 1 IDX$B_DATBKTSZ - data bucket size
349 0411 1 IDX$V_KEY_COMPR - if set, primary key compression is enabled
350 0412 1
351 0413 1 IFAB - address of IFAB
352 0414 1 IFB$V_RU - if set, file is Recovery Unit Journallable
353 0415 1 IFB$V_WRTACC - if set, stream has write access to the file
354 0416 1
355 0417 1 IRAB - address of IRAB
356 0418 1 IRB$L_CURBDB - address of BDB for accessed data bucket
357 0419 1
358 0420 1 REC_ADDR - address of record RMS is positioned to
359 0421 1
360 0422 1 OUTPUT PARAMETERS:
361 0423 1 NONE
362 0424 1
363 0425 1 IMPLICIT OUTPUTS:
364 0426 1
365 0427 1 IRB$L_CURBDB - address of BDB describing primary data bucket
366 0428 1 IRB$V_DEL_SEEN - set if deleted records encountered
367 0429 1 IRB$L_LST_NCMP - updated to last record with a 0 front compressed key

```

```

368 0430 1 | REC_ADDR - address of next non-deleted primary data record
369 0431 1 |
370 0432 1 | ROUTINE VALUE:
371 0433 1 |
372 0434 1 | SUC - found next non-deleted primary data record.
373 0435 1 | EOF - end-of-file encountered before record found.
374 0436 1 | various I/O errors
375 0437 1 |
376 0438 1 | SIDE EFFECTS:
377 0439 1 |
378 0440 1 | SUC - REC_ADDR points to next non-deleted primary data record, and
379 0441 1 | IRB$L_CURBDB contains the address of the BDB describing the
380 0442 1 | primary data bucket.
381 0443 1 | On all errors, no primary data bucket is accessed.
382 0444 1 | If deleted records are encountered the flag IRB$V_DEL_SEEN is set.
383 0445 1 | If RU_DELETED records are encountered, they might have been deleted.
384 0446 1 | If RU_UPDATED records are encountered, they might have been reformatted.
385 0447 1 |
386 0448 1 | --
387 0449 1 |
388 0450 2 | BEGIN
389 0451 2 |
390 0452 2 | BUILTIN
391 0453 2 | AP;
392 0454 2 |
393 0455 2 | EXTERNAL REGISTER
394 0456 2 | COMMON RAB_STR,
395 0457 2 | R_IDX_DFN_STR,
396 0458 2 | R_REC_ADDR_STR;
397 0459 2 |
398 0460 2 | GLOBAL REGISTER
399 0461 2 | COMMON_IO_STR;
400 0462 2 |
401 0463 2 | LOCAL
402 0464 2 | VBN;
403 0465 2 |
404 0466 2 | ! Retrieve the address of the start of the primary data bucket.
405 0467 2 |
406 0468 2 | BDB = .IRAB[IRB$L_CURBDB];
407 0469 2 | BKT_ADDR = .BDB[BDB$S_ADDR];
408 0470 2 |
409 0471 2 | ! If RMS is to start the positioning to the next non-deleted primary data
410 0472 2 | record with the next record, and not the record it is currently
411 0473 2 | positioned to, position to the next record.
412 0474 2 |
413 0475 2 | IF NOT .FLAG
414 0476 2 | THEN
415 0477 2 | BEGIN
416 0478 2 |
417 0479 2 | ! If primary key compression is enabled and the key of the record is
418 0480 2 | zero-front compressed, then save the address of the record as the last
419 0481 2 | record encountered with a zero-front compressed key.
420 0482 2 |
421 0483 2 | IF .IDX_DFN[IDX$V_KEY_COMPR]
422 0484 2 | AND
423 0485 2 | (.REC_ADDR + RMS$REC_OVHD(0)) < 8,8 > EQLU 0
424 0486 2 | THEN

```



```

: 482 0544 5
: 483 0545 5
: 484 0546 5
: 485 0547 4
: 486 0548 5
: 487 0549 5
: 488 0550 5
: 489 0551 5
: 490 0552 5
: 491 0553 5
: 492 0554 5
: 493 0555 5
: 494 0556 5
: 495 0557 5
: 496 0558 5
: 497 0559 5
: 498 0560 5
: 499 0561 5
: 500 0562 5
: 501 0563 5
: 502 0564 5
: 503 0565 5
: 504 0566 5
: 505 0567 5
: 506 0568 5
: 507 0569 5
: 508 0570 5
: 509 0571 4
: 510 0572 4
: 511 0573 4
: 512 0574 4
: 513 0575 4
: 514 0576 4
: 515 0577 4
: 516 0578 4
: 517 0579 4
: 518 0580 4
: 519 0581 4
: 520 0582 3
: 521 0583 3
: 522 0584 3
: 523 0585 4
: 524 0586 4
: 525 0587 4
: 526 0588 4
: 527 0589 4
: 528 0590 4
: 529 0591 4
: 530 0592 4
: 531 0593 4
: 532 0594 4
: 533 0595 4
: 534 0596 4
: 535 0597 4
: 536 0598 4
: 537 0599 4
: 538 0600 4

```

```

: current record as if it actually is deleted, and
: positions to the next record so as to continue the search.
ELSE
  BEGIN
    : If index compression is enabled and the key of the
    : record is zero-front compressed, then save the
    : address of the record as the last record encountered
    : with a zero-front compressed key.
    IF .IDX_DFN[IDX$V_KEY_COMP]
      AND
      (.REC_ADDR + RMSREC_OVHD(0)) < 8,8 > EQLU 0
    THEN
      IRAB[IRB$L_LST_NCMP] = .REC_ADDR;

      IRAB[IRB$V_DEL_SEEN] = 1;
      RMSGETNEXT_REC();
    END

    : RMS was able to reclaim some space from the current
    : record because the Recovery Unit in which the record had
    : been deleted has successfully terminated. RMS will
    : continue the search for a non-deleted primary data record
    : with the next record in the file.
  ELSE
    IF .RECORD_ID NEQU .REC_ADDR[IRC$W_ID]
    THEN
      IRAB[IRB$V_DEL_SEEN] = 1;
    END

    : If the record RMS has positioned to is not deleted, then RMS has
    : found its next non-deleted primary data record, and can return
    : success. Otherwise, set the deleted record seen bit, and position
    : to the very next record in the primary data bucket.
  ELSE
    IF NOT .REC_ADDR[IRC$V_DELETED]
    THEN
      BEGIN
        : If the current non-deleted primary data record has been
        : updated within a Recovery Unit then it maybe re-formatted
        : before it is returned provided the file has been open
        : for write access, and the Recovery Unit in which it was
        : eas updated has completed successfully.
        IF .REC_ADDR[IRC$V_RU_UPDATE]
          AND
          .IFAB[IFB$V_WRTACC]
        THEN
          RMSRU_RECLAIM();

          RETURN RMSSUC();
        END
      END
    END
  END

```

```

539 0601 3 ELSE
540 0602 4 BEGIN
541 0603 4
542 0604 4 ! If index compression is enabled and the key of the record
543 0605 4 ! is zero-front compressed, then save the address of the
544 0606 4 ! record as the last record encountered with a zero-front
545 0607 4 ! compressed key.
546 0608 4
547 0609 4 IF .IDX_DFN[IDX$V_KEY_COMPR]
548 0610 4 AND
549 0611 4 .(.REC_ADDR + RMSREC_OVHD(0))<8,8> EQLU 0
550 0612 4 THEN
551 0613 4 IRAB[IRB$L_LST_NCMP] = .REC_ADDR;
552 0614 4
553 0615 4 IRAB[IRB$V_DEL_SEEN] = 1;
554 0616 4 RMSGETNEXT_REC();
555 0617 4 END;
556 0618 4
557 0619 4 ! If all records in the current record have been exhausted without
558 0620 4 ! find a non-deleted primary data record, and this is the very last
559 0621 4 ! bucket in the horizontal chain, then return an error of end-cf-file.
560 0622 4
561 0623 4 IF .BKT_ADDR[BKT$V_LASTBKT]
562 0624 4 THEN
563 0625 4 BEGIN
564 0626 4 RELEASE (IRAB[IRB$L_CURBDB]);
565 0627 4 RETURN RMSERR(EOF);
566 0628 4 END;
567 0629 4
568 0630 4 ! If this is not the last bucket in the horizontal chain, get the
569 0631 4 ! very next bucket and continue the search for the next non-deleted
570 0632 4 ! primary data bucket in it.
571 0633 4
572 0634 4 VBN = .BKT_ADDR[BKT$L_NXTBKT];
573 0635 4 RELEASE (IRAB[IRB$L_CURBDB]);
574 0636 4
575 0637 4 ! If the file is Recovery Unit Journallable and write accessed, then
576 0638 4 ! make sure the next primary data bucket in the horizontal chain is
577 0639 4 ! exclusively accessed.
578 0640 4
579 0641 4 IF .IFAB[IFB$V_WRTACC]
580 0642 4 AND
581 0643 4 .IFAB[IFB$V_RU]
582 0644 4 THEN
583 0645 4 IRAB[IRB$B_CACHEFLGS] = CSH$M_LOCK;
584 0646 4
585 0647 4 RETURN_ON_ERROR (RMSGETBKT (.VBN, .IDX_DFN[IDX$B_DATBKTSZ] * 512));
586 0648 4
587 0649 4 IRAB[IRB$L_CURBDB] = .BDB;
588 0650 4
589 0651 4 REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
590 0652 4
591 0653 4 ! Make sure the address of the first record in this bucket is placed
592 0654 4 ! in IRB$L_LST_NCMP if primary key compression is enabled.
593 0655 4
594 0656 4 IF .IDX_DFN[IDX$V_KEY_COMPR]
595 0657 4 THEN

```



```

601 0662 1 XSBTTL 'RMSFIND_SIDR'
602 0663 1 ROUTINE RMSFIND_SIDR : RLSLINKAGE =
603 0664 1 ++
604 0665 1
605 0666 1 FUNCTIONAL DESCRIPTION:
606 0667 1
607 0668 1 The function of this routine is to position to a particular SIDR.
608 0669 1
609 0670 1 If the NRP key of reference coincides with the key of reference of the
610 0671 1 current index descriptor, then this SIDR is the current record. In such
611 0672 1 a case the NRP information is used to attempt to position to the
612 0673 1 current SIDR, and if that fails an attempt is made to position to the
613 0674 1 current SIDR by means of a random search of the alternate index. If RMS
614 0675 1 is able to locate the current record, the number of array elements
615 0676 1 preceding the current element is placed in IRBSW_SAVE_POS, and success
616 0677 1 is returned. If RMS is unable to position to the current record because
617 0678 1 it has been deleted, then IRBSW_SAVE_POS is zeroed, and an RMS error
618 0679 1 of record deleted is returned. This status may not represent an actual
619 0680 1 error, and so the SIDR positioned to is also returned. If RMS is unable
620 0681 1 to locate any record, than an error of RNF is returned.
621 0682 1
622 0683 1 If the NRP key of reference is not equal to the key of reference of the
623 0684 1 current index descriptor, then RMS positions to the first SIDR with a
624 0685 1 key greater than or equal to to the key found in keybuffer 2, or to the
625 0686 1 first SIDR with a key equal to the key in keybuffer 2, depending upon
626 0687 1 the IRBSB_SRCHKEY bit settings. If RMS is successful at positioning to
627 0688 1 a SIDR, IRBSW_SAVE_POS will always be zero and a status of success is
628 0689 1 returned; otherwise, a RMS error of record not found is returned.
629 0690 1
630 0691 1 CALLING SEQUENCE:
631 0692 1
632 0693 1 BSBW RMSFIND_SIDR()
633 0694 1
634 0695 1 INPUT PARAMETERS:
635 0696 1 NONE
636 0697 1
637 0698 1 IMPLICIT INPUTS:
638 0699 1
639 0700 1     IDX_DFN - index descriptor of index to search
640 0701 1     IDX$B_DATBKTSZ - size of SIDR bucket in VBNS
641 0702 1     IDX$B_KEYREF - key of reference of index descriptor
642 0703 1     IDX$B_KEYSZ - size of SIDR key
643 0704 1
644 0705 1     IFAB - address of IFAB
645 0706 1     IFBSW_KBUFSZ - size of a keybuffer
646 0707 1     IFBSV_RU - if set, file is Recovery Unit Journallable
647 0708 1     IFBSV_WRTACC - if set, stream has write access to the file
648 0709 1
649 0710 1     IRAB - address of IRAB
650 0711 1     IRBSL_CURBDB - must be zero
651 0712 1     IRBSW_CUR_COUNT - number of elements preceding current element
652 0713 1     IRBSB_CUR_KREF - key of reference of current SIDR record
653 0714 1     IRBSL_KEYBUF - address of stream's five keybuffers
654 0715 1     IRBSW_SIDR_ID - ID of current SIDR's first array element
655 0716 1     IRBSL_SIDR_VBN - VBN of current SIDR's first array element
656 0717 1     IRBSV_SRCHGE - if set, search for GE match
657 0718 1     IRBSV_SRCHGT - if set, search for GT match

```

```

RMS$FIND_SIDR
: 658 0719 1 | IRB$B_STOPLEVEL - level at which to stop search (must be 0)
: 659 0720 1 |
: 660 0721 1 | OUTPUT PARAMETERS:
: 661 0722 1 |     NONE
: 662 0723 1 |
: 663 0724 1 | IMPLICIT OUTPUTS:
: 664 0725 1 |
: 665 0726 1 |     IRB$L_CURBDB - address of SIDR bucket's BDB
: 666 0727 1 |     IRB$W_SAVE_POS - number of elements preceeding the current one
: 667 0728 1 |     REC_ADDR - address of SIDR record
: 668 0729 1 |
: 669 0730 1 | ROUTINE VALUE:
: 670 0731 1 |
: 671 0732 1 |     DEL - if RMS was to position to the current SIDR and found it
: 672 0733 1 |           deleted.
: 673 0734 1 |     RNF - if RMS was unable to find any SIDR to position to and EOF
: 674 0735 1 |           was not encountered.
: 675 0736 1 |     SUC - if the desired SIDR was positioned to.
: 676 0737 1 |     various I/O errors.
: 677 0738 1 |
: 678 0739 1 | SIDE EFFECTS:
: 679 0740 1 |
: 680 0741 1 |     If RMS is successful at positioning to the current SIDR by means of the
: 681 0742 1 |     NRP information, IRB$W_SAVE_POS, REC_ADDR, and IRB$L_CURBDB all
: 682 0743 1 |     have their stated values and a status of SUC is returned.
: 683 0744 1 |     If RMS is unable to successfully position to the current SIDR by means
: 684 0745 1 |     of the NRP information, but is able to position to a record, then
: 685 0746 1 |     REC_ADDR and IRB$L_CURBDB have their assigned values,
: 686 0747 1 |     IRB$W_SAVE_POS is set to 0, and a status of DEL is returned.
: 687 0748 1 |     If RMS is not positioning to the current SIDR and is able to position
: 688 0749 1 |     to some record, then IRB$W_SAVE_POS is set to 0, REC_ADDR and
: 689 0750 1 |     IRB$L_CURBDB have their assigned values, and a status of SUC is
: 690 0751 1 |     returned.
: 691 0752 1 |     If RMS is not positioning to the current SIDR and is unable to position
: 692 0753 1 |     to some record, then IRB$W_SAVE_POS is set to 0, and a status of
: 693 0754 1 |     RNF is returned.
: 694 0755 1 | --
: 695 0756 1 |
: 696 0757 2 | BEGIN
: 697 0758 2 |
: 698 0759 2 | EXTERNAL REGISTER
: 699 0760 2 |     COMMON RAB STR,
: 700 0761 2 |     R_IDX_DFN STR,
: 701 0762 2 |     R_REC_ADDR;
: 702 0763 2 |
: 703 0764 2 | LOCAL
: 704 0765 2 |     FLAGS : BLOCK[1];
: 705 0766 2 |
: 706 0767 2 | MACRO
: 707 0768 2 |     FIRST_TIME = 0,0,1,0 %;
: 708 0769 2 |     NRP = 0,1,1,0 %;
: 709 0770 2 |
: 710 0771 2 | ! Initialize some variables.
: 711 0772 2 | !
: 712 0773 2 |     FLAGS = 0;
: 713 0774 2 |     FLAGS[FIRST_TIME] = 1;
: 714 0775 2 |

```

```

: 715 0776 2 IRAB[IRBSW_SAVE_POS] = 0;
: 716 0777 2
: 717 0778 2 ! If the index descriptor is for the current key of reference, then the
: 718 0779 2 ! NRP information will be used in the positioning.
: 719 0780 2
: 720 0781 3 IF (.IDX_DFN[IDX$B_KEYREF] EQLU .IRAB[IRBSB_CUR_KREF])
: 721 0782 2 THEN
: 722 0783 2     FLAGS[NRP] = 1;
: 723 0784 2
: 724 0785 2 ! We may have to make two circuits through roughly the same code;
: 725 0786 2 ! therefore, setup the circumstances for doing so - ie simulate a GOTO.
: 726 0787 2 ! This loop can only be exited by returning to the calling routine.
: 727 0788 2
: 728 0789 2 WHILE 1
: 729 0790 2 DO
: 730 0791 2     BEGIN
: 731 0792 3
: 732 0793 3     GLOBAL REGISTER
: 733 0794 3         COMMON_IO_STR;
: 734 0795 3
: 735 0796 3     LOCAL
: 736 0797 3         SEARCH_RESULT;
: 737 0798 3
: 738 0799 3     ! If the next record positioning formation is to be used, and this is
: 739 0800 3     ! the first time through the loop, then use the next record context in
: 740 0801 3     ! the attempt to positioning to the desired SIDR.
: 741 0802 3
: 742 0803 3     IF .FLAGS[NRP] AND .FLAGS[FIRST_TIME]
: 743 0804 3     THEN
: 744 0805 4         BEGIN
: 745 0806 4
: 746 0807 4             ! All SIDR buckets are exclusively accessed in case space
: 747 0808 4             ! reclamation will be required.
: 748 0809 4
: 749 0810 4             IRAB[IRBSB_CACHEFLGS] = CSH$M_LOCK;
: 750 0811 4
: 751 0812 4             ! First, access the bucket hopefully containing the current record
: 752 0813 4             ! using the next record context information.
: 753 0814 4
: 754 P 0815 4             RETURN_ON_ERROR (RMSGETBKT (.IRAB[IRBSL_CUR_VBN],
: 755 0816 4                 .IDX_DFN[IDX$B_DATBKTSZ]*512));
: 756 0817 4             IRAB[IRBSL_CURBDB] = .BDB;
: 757 0818 4
: 758 0819 4             ! Next, do a search through this bucket looking for a SIDR with a
: 759 0820 4             ! key value the same as that of the current record, the key of
: 760 0821 4             ! which maybe found in keybuffer 2.
: 761 0822 4
: 762 0823 4             BKT_ADDR = .BBLOCK[.IRAB[IRBSL_CURBDB], BDB$SL_ADDR];
: 763 0824 4             REC_ADDR = .BKT_ADDR + BKT$C_OVERHDSZ;
: 764 0825 4
: 765 0826 4             SEARCH_RESULT = RMSRCH_BY_KEY();
: 766 0827 4             END
: 767 0828 4
: 768 0829 4             ! If the next record positioning information can not be used to
: 769 0830 4             ! position to the desired SIDR either because it is for the wrong key
: 770 0831 4             ! of reference, or because this is not the first time through this
: 771 0832 4             ! loop, then perform a random search of the alternate index using as

```

772	0833	4
773	0834	4
774	0835	3
775	0836	4
776	0837	4
777	0838	4
778	0839	4
779	0840	4
780	0841	4
781	0842	4
782	0843	4
783	0844	4
784	0845	4
785	0846	3
786	0847	3
787	0848	3
788	0849	3
789	0850	3
790	0851	3
791	0852	3
792	0853	3
793	0854	3
794	0855	3
795	0856	3
796	0857	3
797	0858	3
798	0859	3
799	0860	4
800	0861	4
801	0862	4
802	0863	4
803	0864	4
804	0865	4
805	0866	4
806	0867	4
807	0868	4
808	0869	4
809	0870	4
810	0871	4
811	0872	3
812	0873	3
813	0874	3
814	0875	3
815	0876	3
816	0877	3
817	0878	3
818	0879	3
819	0880	3
820	0881	3
821	0882	3
822	0883	3
823	0884	3
824	0885	4
825	0886	4
826	0887	4
827	0888	4
828	0889	4

```

! the search key the key in keybuffer 2.
ELSE
  BEGIN
    RETURN_ON_ERROR (RMS$SEARCH_TREE());

    ! Determine whether the SIDR positioned to has a key greater
    ! than or equal to or less than the key in keybuffer 2.
    SEARCH_RESULT = RMS$COMPARE_REC (KEYBUF_ADDR(2),
                                     .IDX_DFN[IDX$B_KEYSZ],
                                     0);

  END;

! The SEARCH_RESULT can be greater-than only if RMS is currently
! positioning using the NRP context information, and the key in
! keybuffer 2 is greater than the key of every SIDR in the current
! bucket. In such a situation, RMS knows that the SIDR it was looking
! for has moved since it was accessed by this process. Therefore, this
! SIDR could not have been in a continuation bucket, and if its still
! in the ISAM file it can be positioned to by performing a random
! search of the alternate key index using the record's key value after
! releasing the current SIDR bucket.
IF .SEARCH_RESULT GTR 0
THEN
  BEGIN
    FLAGS[FIRST TIME] = 0;
    RELEASE (IRAB[IRBSL_CURBDB]);
  END

! The SEARCH_RESULT can be less-than only if RMS was unable to find
! a SIDR record with a key equal to the key in keybuffer 2 either in
! the current bucket (if the NRP context information was being
! utilized) or anywhere in the alternate index (if it was not). Under
! such circumstances, the SIDR RMS has positioned to, which has a key
! greater than the key in keybuffer 2, is the desired SIDR.
ELSE
  IF .SEARCH_RESULT LSS 0
  THEN
    ! If RMS had been using the NRP context information to position
    ! to the current SIDR then this means that the current SIDR has
    ! been deleted. To indicate this IRBSW_SAVE_POS is left at
    ! zero and an RMS error of record delete is returned although
    ! whether this is an error or not depends upon the context in
    ! which this routine is called.
    IF .FLAGS[NRP]
    THEN
      RETURN RMSERR(DEL)

    ! If the NRP context information had never been used, then this
    ! SIDR represents the desired SIDR, and a successful status is
    ! returned.

```

```

829 0890 4
830 0891 3
831 0892 4
832 0893 4
833 0894 4
834 0895 4
835 0896 4
836 0897 3
837 0898 3
838 0899 3
839 0900 3
840 0901 3
841 0902 3
842 0903 3
843 0904 3
844 0905 3
845 0906 3
846 0907 3
847 0908 3
848 0909 4
849 0910 4
850 0911 4
851 0912 4
852 0913 4
853 0914 4
854 0915 4
855 0916 4
856 0917 4
857 0918 4
858 0919 4
859 0920 4
860 0921 4
861 0922 4
862 0923 4
863 0924 4
864 0925 4
865 0926 4
866 0927 5
867 0928 4
868 0929 5
869 0930 4
870 0931 5
871 0932 5
872 0933 5
873 0934 5
874 0935 5
875 0936 5
876 0937 5
877 0938 5
878 0939 5
879 0940 5
880 0941 5
881 0942 5
882 0943 5
883 0944 4
884 0945 4
885 0946 4

```

```

ELSE
    RETURN RMSSUC()
! RMS has successfully positioned to a SIDR with a key exactly
! matching the key to be found in keybuffer 2.
ELSE
! If the SIDR RMS is to retrieve is in fact the current record,
! then even though RMS has been able to position to a SIDR
! with the same key as the current record, this SIDR may not be
! the current record. To determine this, the RFA pointer of
! the found SIDR's first array element must be compared with
! the RFA pointer of the first element of the current record
! which is part of the next record context found in the IRAB.
IF .FLAGS[NRP]
THEN
    BEGIN
        LOCAL
            FIRST_VBN,
            FIRST_ID;

! Retrieve the RFA pointer of the SIDR's first element.
RMSSIDR_FIRST (1;FIRST_VBN, FIRST_ID);

! If the RFA pointer of the SIDR's first element matches
! the RFA pointer of the current SIDR's first element
! saved as part of the next record positioning context,
! then RMS has been able to successfully position to the
! current record. Load the number of array elements
! preceding the current element in to IRBSW_SAVE_POS and
! return success.
IF (.FIRST_VBN EQLU .IRAB[IRBSL_SIDR_VBN])
AND
(.FIRST_ID EQLU .IRAB[IRBSW_SIDR_ID])
THEN
    BEGIN
        IRAB[IRBSW_SAVE_POS] = .IRAB[IRBSW_CUR_COUNT];
        RETURN RMSSUC();
    END

! If either the VBN or the ID of the first element of the
! current record and the first element of the SIDR RMS has
! positioned to do not match, then the current SIDR has
! been deleted, IRBSW_SAVE_POS is left at 0, and a RMS
! error of record deleted is returned (although whether
! this is an error or not depends upon the context in
! which this routine is called.
ELSE
    RETURN RMSERR(DEL);
END

```

```

: 886 0947 4
: 887 0948 4
: 888 0949 4
: 889 0950 4
: 890 0951 4
: 891 0952 4
: 892 0953 3
: 893 0954 3
: 894 0955 3
: 895 0956 2
: 896 0957 2
: 897 0958 2
: 898 0959 1

```

```

: If RMS is not asked to specifically retrieve the current
: record, then this SIDR with a key equal to the key in
: keybuffer 2 is the desired SIDR, and a status of success
: is returned.

```

```

ELSE
RETURN RMSSUC();

```

END;

```

END:
L1:0789

```

! { of routine }

```

: INFO#212
: Null expression appears in value-required context

```

			3C	BB	00000	RMSFIND_SIDR:			
				53	D4	00002	PUSHR	#^M<R2,R3,R4,R5>	0663
				01	88	00004	CLRL	FLAGS	0773
	53			A9	B4	00007	BISB2	#1, FLAGS	0774
		00C3	76	A7	91	0000A	CLRW	118(IRAB)	0776
			21				CMPB	33(IDX_DFN), 195(IRAB)	0781
				03	12	00010	BNEQ	1\$	
	53			02	88	00012	BISB2	#2, FLAGS	0783
31				01	E1	00015	BBC	#1, FLAGS, 2\$	0803
	53			53	E9	00019	BLBC	FLAGS, 2\$	
	2E			01	90	0001C	MOVB	#1, 64(IRAB)	0810
	40			A9			MOVZBL	23(IDX_DFN), R0	0816
	50		17	A7	9A	00020	ASHL	#9, R0, -(SP)	
7E				09	78	00024	PUSHL	168(IRAB)	
			00A8	C9	DD	00C28	BSBW	RMSGETBKT	
				0000G	30	0002C	ADDL2	#8, SP	
	5E			08	C0	0002F	BLBC	STATUS, 8\$	
	7E			50	E9	00032	MOVL	BDB, 32(IRAB)	0817
	20			54	D0	00035	MOVL	32(IRAB), R0	0823
	50		20	A9	D0	00039	MOVL	24(R0), BKT_ADDR	
	55		18	A0	D0	0003D	MOVAB	14(R5), REC_ADDR	0824
	56		0E	A5	9E	00041	BSBW	RMSRCH_BY_KEY	0826
				0000G	30	00U45	BRB	3\$	0803
				1B	11	00048	BSBW	RMSSEARCH_TREE	0838
	63			0000G	30	0004A	BLBC	STATUS, 8\$	
				50	E9	0004D	CLRL	-(SP)	0843
	7E			7E	D4	00050	MOVZBL	32(IDX_DFN), -(SP)	0844
	50		20	A7	9A	00052	MOVZWL	180(IFAB), R0	0843
			00B4	CA	3C	00056	PUSHAB	@96(IRAB)[R0]	
			60	B940	9F	0005B	BSBW	RMSCOMPARE_REC	
				0000G	30	0005F	ADDL2	#12, SP	
	5E			0C	C0	00062	TSTL	SEARCH_RESULT	0858
				50	D5	00065	BLEQ	4\$	
				14	15	00067	BICB2	#1, FLAGS	0861
	53			01	8A	00069	MOVL	32(IRAB), BDB	0862
	54		20	A9	D0	0006C	CLRL	32(IRAB)	
			20	A9	D4	00070	CLRL	-(SP)	
				7E	D4	00073			

				0000G	30	00075		BSBW	RMSRLSBKT		
			5E	04	C0	00078		ADDL2	#4, SP		
				98	11	0007B		BRB	1\$		0858
				06	18	0007D	4\$:	BGEQ	5\$		0873
	2D		53	01	E1	0007F		BBC	#1, FLAGS, 7\$		0883
				24	11	00083		BRB	6\$		0885
	27		53	01	E1	00085	5\$:	BBC	#1, FLAGS, 7\$		09C7
				01	DD	00089		PUSHL	#1		0917
				00C0G	30	0008B		BSBW	RMS\$SDR_FIRST		
			5E	04	C0	0008E		ADDL2	#4, SP		
		00B4	C9	51	D1	00091		CMPL	FIRST_VBN, 180(IRAB)		0927
				11	12	00096		BNEQ	6\$		
52	00BE	C9	10	08	12	00098		CMPZV	#0, #16, 190(IRAB), FIRST_ID		0929
				08	12	0009F		BNEQ	6\$		
		76	A9	00C0	C9	000A1		MOVW	192(IRAB), 118(IRAB)		0932
				07	11	000A7		BRB	7\$		0945
			50	8262	8F	000A9	6\$:	MOVZWL	#33378, R0		
				03	11	000AE		BRB	8\$		
			50		01	000B0	7\$:	MOVL	#1, R0		0954
					3C	000B3	8\$:	POPR	#^M<R2,R3,R4,R5>		0959
					05	000B5		RSB			

; Routine Size: 182 bytes, Routine Base: RMSRMS3 + 0129

```

: 900
: 901
: 902
: 903
: 904
: 905
: 906
: 907
: 908
: 909
: 910
: 911
: 912
: 913
: 914
: 915
: 916
: 917
: 918
: 919
: 920
: 921
: 922
: 923
: 924
: 925
: 926
: 927
: 928
: 929
: 930
: 931
: 932
: 933
: 934
: 935
: 936
: 937
: 938
: 939
: 940
: 941
: 942
: 943
: 944
: 945
: 946
: 947
: 948
: 949
: 950
: 951
: 952
: 953
: 954
: 955
: 956

```

```

0960 1 %SBTTL 'RM$POS_SEQ'
0961 1 GLOBAL ROUTINE RM$POS_SEQ : RL$RABREG_67 =
0962 1
0963 1
0964 1
0965 1
0966 1
0967 1
0968 1
0969 1
0970 1
0971 1
0972 1
0973 1
0974 1
0975 1
0976 1
0977 1
0978 1
0979 1
0980 1
0981 1
0982 1
0983 1
0984 1
0985 1
0986 1
0987 1
0988 1
0989 1
0990 1
0991 1
0992 1
0993 1
0994 1
0995 1
0996 1
0997 1
0998 1
0999 1
1000 1
1001 1
1002 1
1003 1
1004 1
1005 1
1006 1
1007 1
1008 1
1009 1
1010 1
1011 1
1012 1
1013 1
1014 1
1015 1
1016 1

```

!++

FUNCTIONAL DESCRIPTION:

This routines responsibility is to direct RMS's positioning to the next primary data record to be returned. In some cases the next primary data record will be the current primary data record (if IRBSV_SKIP_NEXT is not set and the current record is not deleted), but in most cases the next primary data record will in fact be the first non-deleted primary data record RMS can position to after positioning to the current primary data record or SIDR array element.

This routine handles the positioning itself if the primary key is the current key of reference, and it also directs the positioning to the very first SIDR if the current index is an alternate index, and this is the first positioning attempt after a \$CONNECT or \$REWIND. However, this routine handles all subsequent SIDR positionings by calling RM\$FIND_SIDR to position to the current SIDR and RM\$SEARCH_SIDR to position to the current array within the SIDR.

RMS's first task is to position to the current record (to the current SIDR array element if the current record is a SIDR). One basic assumption is made by this routine, and that is that if RMS is positioning by the primary key of reference and duplicate primary keys ARE allowed then this process's current record MUST be present in the file. This is because the only time the space held by a record can be reclaimed on that record's deletion is if duplicate primary keys are not allowed in the file. The same can not be said to be true for SIDRs. If every element in a SIDR array is deleted it will be possible to delete the entire SIDR regardless of whether it is any process's current record or not.

Once RMS has positioned to the current record, it can position to the primary data record to be returned (which may in fact be the current primary data record). This positioning is accomplished by this routine, if the primary key is the current key of reference, or by RM\$SEARCH_SIDR if it is not.

The algorithm used by this routine in positioning to the current record when the primary key is the key of reference is as follows:

- 1) If this is the first time a positioning is attempted after a \$CONNECT or \$REWIND there is no current record so position to the very first non-deleted primary data record in the file.
- 2) Otherwise, the current record information is used to access the current bucket and search for the record in there.
- 3) If the current record is not found in the current bucket and the VBN of the current primary data record and current bucket differ, then attempt to find the current primary data record utilizing its RFA address.
- 4) If RMS is still unable to find the current record it must have been

```

: 957 1017 1 | deleted; therefore RMS performs a random search of the index either
: 958 1018 1 | to locate the deleted record (just marked deleted) or to find the
: 959 1019 1 | next record (prologue 3 files not allowing duplicate primary keys
: 960 1020 1 | only).
: 961 1021 1 |
: 962 1022 1 | a) If duplicates are allowed the search is for the first record with
: 963 1023 1 | an identical key. A search is then made of all primary data
: 964 1024 1 | records with this key value until the current record, which is
: 965 1025 1 | guaranteed to be in the file, is found.
: 966 1026 1 |
: 967 1027 1 | b) If duplicates are not allowed then the search is for the first
: 968 1028 1 | record with a key greater than or equal to the key of the
: 969 1029 1 | current record. This is because if duplicates are not allowed
: 970 1030 1 | a greater-than or equal search will position RMS to the first
: 971 1031 1 | record of identical or greater key value, and the next
: 972 1032 1 | non-deleted record, (including the one directly positioned to)
: 973 1033 1 | is the record RMS wants to return.
: 974 1034 1 |
: 975 1035 1 | CALLING SEQUENCE:
: 976 1036 1 |     BSBW RM$POS_SEQ()
: 977 1037 1 |
: 978 1038 1 | INPUT PARAMETERS:
: 979 1039 1 |     NONE
: 980 1040 1 |
: 981 1041 1 | IMPLICIT INPUTS:
: 982 1042 1 |
: 983 1043 1 |     IDX_DFN - address of index descriptor
: 984 1044 1 |     -IDXSB_DATABKTSZ - size of data bucket
: 985 1045 1 |     IDXSB_KEYREF - key of reference
: 986 1046 1 |
: 987 1047 1 |     IFAB - address of IFAB
: 988 1048 1 |     IFBSW_KBUFSZ - keybuffer size
: 989 1049 1 |     IFBSV_RU - if set, file is Recovery Unit Journallable
: 990 1050 1 |     IFBSV_WRTACC - if set, stream has write access to the file
: 991 1051 1 |
: 992 1052 1 |     IRAB - address of IRAB
: 993 1053 1 |     IRBSW_CUR_ID - ID of current record
: 994 1054 1 |     IRBSL_CUR_VBN - VBN of current record
: 995 1055 1 |     IRBSL_KEYBUF - address of keybuffers
: 996 1056 1 |     IRBSV_SKIP_NEXT - if set, skip current record and return next
: 997 1057 1 |     IRBSW_POS_ID - ID of user data record for NRP positioning
: 998 1058 1 |     IRBSL_POS_VBN - VBN of user data record for NRP positioning
: 999 1059 1 |
: 1000 1060 1 | OUTPUT PARAMETERS:
: 1001 1061 1 |     NONE
: 1002 1062 1 |
: 1003 1063 1 | IMPLICIT OUTPUTS:
: 1004 1064 1 |
: 1005 1065 1 |     IRBSL_CURBDB - address of the BDB describing the primary data bucket
: 1006 1066 1 |     IRBSV_DEL_SEEN - deleted records seen between current and next record
: 1007 1067 1 |     IRBSW_FIRST_ID - ID of the next SIDR's first element's RRV pointer
: 1008 1068 1 |     IRBSL_FIRST_VBN - VBN of the next SIDR's first element's RRV pointer
: 1009 1069 1 |     IRBSW_RFA_ID - ID of the next record (primary data only)
: 1010 1070 1 |     IRBSL_RFA_VBN - VBN of the next record (SIDR or primary data)
: 1011 1071 1 |     IRBSW_SAVE_POS - number of elements preceding next SIDR array element
: 1012 1072 1 |     REC_ADDR - address of next primary data record
: 1013 1073 1 |

```

```

: 1014 1074 1 | ROUTINE VALUE:
: 1015 1075 1 |
: 1016 1076 1 | EOF - end-of-file encountered before record to be returned is found.
: 1017 1077 1 | SUC - found non-deleted primary data record to return.
: 1018 1078 1 | BUG - serious internal error precluded positioning to a record.
: 1019 1079 1 | various I/O errors
: 1020 1080 1 |
: 1021 1081 1 | SIDE EFFECTS:
: 1022 1082 1 |
: 1023 1083 1 | SUC - if the primary key is the key of reference, IRB$RFA_VBN and
: 1024 1084 1 | IRB$RFA_ID have their assigned values, and if not,
: 1025 1085 1 | IRB$RFA_VBN, IRB$FIRST_VBN, IRB$FIRST_ID, and
: 1026 1086 1 | IRB$SAVE_POS have their assign values. In both cases, REC_ADDR,
: 1027 1087 1 | IRB$DEL_SEEN, and IRB$CURBDB have their assigned values.
: 1028 1088 1 | In all other cases, these fields have indeterminate values and all
: 1029 1089 1 | bucket locks have been released.
: 1030 1090 1 | AP IS TRASHED
: 1031 1091 1 | IRB$STOPLEVEL is 0
: 1032 1092 1 | IRB$SRCHFLGS is of indeterminate status
: 1033 1093 1 |
: 1034 1094 1 | --
: 1035 1095 1 |
: 1036 1096 2 | BEGIN
: 1037 1097 2 |
: 1038 1098 2 | EXTERNAL REGISTER
: 1039 1099 2 | COMMON RAB_STR,
: 1040 1100 2 | R_IDX_DFN_STR,
: 1041 1101 2 | R_REC_ADDR_STR;
: 1042 1102 2 |
: 1043 1103 2 |
: 1044 1104 2 | ! Initialize some search control fields.
: 1045 1105 2 |
: 1046 1106 2 | IRAB[IRB$STOPLEVEL] = 0;
: 1047 1107 2 | IRAB[IRB$SRCHFLAGS] = 0;
: 1048 1108 2 |
: 1049 1109 2 | ! If this is the very first time RMS is positioning after a $CONNECT or
: 1050 1110 2 | $REWIND then position to the very first non-RRV record in the very first
: 1051 1111 2 | SIDR/primary data bucket.
: 1052 1112 2 |
: 1053 1113 3 | IF (.IRAB[IRB$CUR_VBN] EQLU 0)
: 1054 1114 2 | AND
: 1055 1115 3 | (.IRAB[IRB$POS_VBN] EQLU 0)
: 1056 1116 2 | THEN
: 1057 1117 3 | BEGIN
: 1058 1118 3 |
: 1059 1119 3 | IRAB[IRB$FIRST_TIM] = 1;
: 1060 1120 3 | RET_ON_ERR_EOF(RM$SEARCH_TREE(), IRAB[IRB$FIRST_TIM] = 0);
: 1061 1121 3 |
: 1062 1122 3 | ! If RM$SEARCH_TREE positioned RMS to an empty bucket, position to the
: 1063 1123 3 | first non-empty bucket or until the end-of-file is encountered.
: 1064 1124 3 |
: 1065 1125 3 | WHILE 1
: 1066 1126 3 | DO
: 1067 1127 4 | BEGIN
: 1068 1128 4 |
: 1069 1129 4 | GLOBAL REGISTER
: 1070 1130 4 | COMMON_IO_STR;

```

```

: 1071 1131 4
: 1072 1132 4
: 1073 1133 4
: 1074 1134 4
: 1075 1135 4
: 1076 1136 4
: 1077 1137 4
: 1078 1138 4
: 1079 1139 4
: 1080 1140 4
: 1081 1141 4
: 1082 1142 4
: 1083 1143 6
: 1084 1144 6
: 1085 1145 6
: 1086 1146 5
: 1087 1147 5
: 1088 1148 4
: 1089 1149 4
: 1090 1150 4
: 1091 1151 4
: 1092 1152 4
: 1093 1153 4
: 1094 1154 4
: 1095 1155 4
: 1096 1156 4
: 1097 1157 5
: 1098 1158 5
: 1099 1159 5
: 1100 1160 4
: 1101 1161 4
: 1102 1162 4
: 1103 1163 4
: 1104 1164 4
: 1105 1165 4
: 1106 1166 4
: 1107 1167 4
: 1108 1168 4
: 1109 1169 4
: 1110 1170 4
: 1111 1171 4
: 1112 1172 4
: 1113 1173 5
: 1114 1174 5
: 1115 1175 5
: 1116 1176 4
: 1117 1177 4
: 1118 1178 4
: 1119 1179 4
: 1120 1180 4
: 1121 1181 4
: 1122 1182 4
: 1123 1183 4
: 1124 1184 4
: 1125 1185 3
: 1126 1186 3
: 1127 1187 3

```

```

LOCAL
  VBN;

BKT_ADDR = .BBLOCK[.IRAB[IRB$$_CURBDB], BDB$$_ADDR];

! If RMS has positioned to a RRV in a primary data bucket, or if
! the bucket is empty, then RMS must release this bucket, and access
! the next one in the horizontal chain. Otherwise, RMS starts its
! search for the first non-deleted primary data record with the
! first record in this bucket.

IF NOT ((.IDX_DFN[IDX$$_KEYREF] EQLU 0
        AND
        .REC_ADDR[IRC$$_RRV])
       OR
       (.BKT_ADDR[BKT$$_FREESPACE] LEQU BKT$$_OVERHDSZ))
THEN
  EXITLOOP;

! If this is the last data bucket in the horizontal chain then
! every single bucket is empty. End the search for the first
! non-empty bucket and return end-of-file.

IF .BKT_ADDR[BKT$$_LASTBKT]
THEN
  BEGIN
    RELEASE(IRAB[IRB$$_CURBDB]);
    RETURN RMSERR(E0F);
  END;

! Save the VBN of the next bucket in the horizontal chain, release
! the current bucket, and then access the next bucket by its VBN.

VBN = .BKT_ADDR[BKT$$_NEXTBKT];
RELEASE(IRAB[IRB$$_CURBDB]);

! If the file is Recovery Unit Journallable and write accessed or
! the positioning is being done by alternate key of reference, then
! make sure the next data bucket in the horizontal chain is
! exclusively accessed.

IF (.IFAB[IFB$$_WRTACC]
   AND
   .IFAB[IFB$$_RU])
  OR
  .IDX_DFN[IDX$$_KEYREF] NEQU 0
THEN
  IRAB[IRB$$_CACHEFLGS] = CSH$$_LOCK;

RETURN_ON_ERROR (RMSGETBKT(.VBN, .IDX_DFN[IDX$$_DATBKTSZ]*512));

IRAB[IRB$$_CURBDB] = .BDB;
REC_ADDR = .BKT_ADDR + BKT$$_OVERHDSZ;
END;

! When the first non-empty bucket has been encountered and REC_ADDR

```

```

: 1128      1188      3      ! is setup to point to the first record in that bucket, then clear the
: 1129      1189      3      ! first time flag.
: 1130      1190      3      !
: 1131      1191      3      ! IRAB[IRBSV_FIRST_TIM] = 0;
: 1132      1192      3      !
: 1133      1193      3      ! If the key of reference is the primary key, then RMS must find the
: 1134      1194      3      ! first non-deleted primary data record in the file. If this is an
: 1135      1195      3      ! alternate index, RMS does not need to do this positioning because
: 1136      1196      3      ! that will be done when RMS positions to the first non-deleted
: 1137      1197      3      ! primary data record.
: 1138      1198      3      !
: 1139      1199      3      ! IF .IDX_DFN[IDX$B_KEYREF] EQL 0
: 1140      1200      3      ! THEN
: 1141      1201      4      !     RETURN_ON_ERROR (RMSFIND_NONDEL (1))
: 1142      1202      4      !
: 1143      1203      4      ! If the key of reference is not the primary key, then RMS is to
: 1144      1204      4      ! start its search for the first non-deleted primary data record in
: 1145      1205      4      ! the file with the very first array element of the SIDR it has
: 1146      1206      4      ! positioned to.
: 1147      1207      4      !
: 1148      1208      3      ! ELSE
: 1149      1209      3      !     IRAB[IRBSW_SAVE_POS] = 0;
: 1150      1210      3      ! END
: 1151      1211      3      !
: 1152      1212      3      ! This is not the first time RMS has sequentially accessed the file since
: 1153      1213      3      ! a $CONNECT, or $REWIND was done. If RMS is positioning by means of an
: 1154      1214      3      ! alternate key of reference, then all of the positioning to the current
: 1155      1215      3      ! SIDR and the current array element within that SIDR is done by the
: 1156      1216      3      ! routine RMSFIND_SIDR.
: 1157      1217      3      !
: 1158      1218      2      ! ELSE
: 1159      1219      2      !     IF .IDX_DFN[IDX$B_KEYREF] NEQ 0
: 1160      1220      2      !     THEN
: 1161      1221      2      !         BEGIN
: 1162      1222      2      !
: 1163      1223      2      !         LOCAL
: 1164      1224      2      !             STATUS;
: 1165      1225      2      !
: 1166      1226      2      !         IRAB[IRBSV_SRCHGE] = 1;
: 1167      1227      2      !
: 1168      1228      2      !         ! If some serious error occurred during RMS's positioning to the
: 1169      1229      2      !         ! current SIDR, then immediately return the error. However, if
: 1170      1230      2      !         ! RMS has successfully positioned to the current SIDR then
: 1171      1231      2      !         ! IRBSW_SAVE_POS contains the number of array elements preceding
: 1172      1232      2      !         ! the current SIDR array element.
: 1173      1233      2      !         !
: 1174      1234      2      !         ! IF NOT ((STATUS = RMSFIND_SIDR())
: 1175      1235      2      !         !     OR
: 1176      1236      2      !         !     (.STATUS<0,16> EQL RMSERR(DEL)))
: 1177      1237      2      !         ! THEN
: 1178      1238      2      !         !
: 1179      1239      2      !         !     ! If the current SIDR was the last SIDR in the file, and it
: 1180      1240      2      !         !     ! has been deleted, return EOF indicating that the end of the
: 1181      1241      2      !         !     ! file has been encountered; otherwise, return the actual error.
: 1182      1242      2      !         !
: 1183      1243      2      !         !     IF .STATUS<0,16> EQLU RMSERR(RNF)
: 1184      1244      2      !         !     THEN

```

B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
[
,
-
.
/

```

: 1185      1245      4
: 1186      1246      4
: 1187      1247      4
: 1188      1248      4
: 1189      1249      4
: 1190      1250      4
: 1191      1251      4
: 1192      1252      4
: 1193      1253      4
: 1194      1254      4
: 1195      1255      4
: 1196      1256      4
: 1197      1257      4
: 1198      1258      4
: 1199      1259      4
: 1200      1260      4
: 1201      1261      4
: 1202      1262      4
: 1203      1263      4
: 1204      1264      4
: 1205      1265      4
: 1206      1266      4
: 1207      1267      4
: 1208      1268      4
: 1209      1269      4
: 1210      1270      4
: 1211      1271      4
: 1212      1272      4
: 1213      1273      4
: 1214      1274      4
: 1215      1275      4
: 1216      1276      4
: 1217      1277      4
: 1218      1278      4
: 1219      1279      4
: 1220      1280      4
: 1221      1281      4
: 1222      1282      4
: 1223      1283      4
: 1224      1284      4
: 1225      1285      4
: 1226      1286      4
: 1227      1287      4
: 1228      1288      4
: 1229      1289      4
: 1230      1290      4
: 1231      1291      4
: 1232      1292      4
: 1233      1293      4
: 1234      1294      4
: 1235      1295      4
: 1236      1296      4
: 1237      1297      4
: 1238      1298      4
: 1239      1299      4
: 1240      1300      4
: 1241      1301      4

        RETURN RMSERR(EOF)
      ELSE
        RETURN .STATUS;

      ! If RMS is successful at positioning to the current SIDR then
      ! setup to return the next record provided RMS is to return the
      ! next record (IRBSV_SKIP_NEXT is set).
      IF .STATUS AND .IRAB[IRBSV_SKIP_NEXT]
      THEN
        IRAB[IRBSW_SAVE_POS] = .IRAB[IRBSW_SAVE_POS] + 1;
      END

      ! If this is not the first time sequentially accessing a record after
      ! a $REWIND or $CONNECT, and RMS is to position by the primary key of
      ! reference, then RMS's strategy is to first position to the current
      ! record which must be present somewhere in the file if it allows
      ! duplicate primary keys. RMS will return either its position or that
      ! of the next non-deleted primary data record which follows it based
      ! upon whether or not it found the current record to be deleted, and
      ! whether it was to return the current or the next record.
    ELSE
      BEGIN
        FIELD
          POS_FLAGS =
            SET
              FND_BY_RRV = [0,0,1,0],
              FND_BY_SRCH = [0,1,1,0]
            TES;

        LOCAL
          FLAG      : BLOCK[1,BYTE]
                    FIELD (POS_FLAGS);

        FLAG = 0;

        ! RMS will use the VBN and ID of the current record positioning
        ! information to attempt to position directly to the current
        ! primary data record.
        BEGIN
          BUILTIN
            AP;

          LOCAL
            STATUS;

          ! If the file is Recovery Unit Journallable and write accessed, then
          ! make sure the current primary data bucket exclusively accessed.
          IF .IFAB[!FBSV_WRTACC]
            AND
              .IFAB[IFBSV_RU]
          THEN

```

```

: 1242
: 1243
: 1244
: 1245
: 1246
: 1247
: 1248
: 1249
: 1250
: 1251
: 1252
: 1253
: 1254
: 1255
: 1256
: 1257
: 1258
: 1259
: 1260
: 1261
: 1262
: 1263
: 1264
: 1265
: 1266
: 1267
: 1268
: 1269
: 1270
: 1271
: 1272
: 1273
: 1274
: 1275
: 1276
: 1277
: 1278
: 1279
: 1280
: 1281
: 1282
: 1283
: 1284
: 1285
: 1286
: 1287
: 1288
: 1289
: 1290
: 1291
: 1292
: 1293
: 1294
: 1295
: 1296
: 1297
: 1298

```

```

1302 4
1303 4
1304 4
1305 4
1306 4
1307 4
1308 4
1309 4
1310 5
1311 5
1312 6
1313 5
1314 5
1315 4
1316 4
1317 4
1318 4
1319 4
1320 4
1321 4
1322 4
1323 4
1324 4
1325 4
1326 4
1327 5
1328 5
1329 5
1330 5
1331 5
1332 5
1333 5
1334 5
1335 5
1336 5
1337 5
1338 5
1339 5
1340 5
1341 5
1342 5
1343 5
1344 5
1345 5
1346 5
1347 5
1348 5
1349 5
1350 5
1351 5
1352 5
1353 5
1354 5
1355 5
1356 5
1357 5
1358 5

```

```

IRAB[IRB$B_CACHEFLGS] = CSH$M_LOCK;
AP = .IRAB[IRB$W_CUR_ID];
STATUS = RMS$FIND_BY_RFA (.IRAB[IRB$L_CUR_VBN]);
! On any errors other than record not found or record deleted
! return immediately. These will be I/O errors.
IF NOT (.STATUS
      OR
      (.STATUS<0,16> EQL RMSERR(RNF))
      OR
      (.STATUS<0,16> EQL RMSERR(DEL)))
THEN
  RETURN .STATUS;
! If RMS is unable to find any record through use of the
! current record information, or if it finds a RRV then this
! indicates that the current record must have moved, or may have
! been deleted.
IF NOT .STATUS
  OR
  .REC_ADDR[IRCSV_RRV]
THEN
  BEGIN
    GLOBAL REGISTER
    COMMON_IO_STR;
    ! If the VBN of the current record is not the same as the
    ! VBN of the current record's RFA address, then RMS has no
    ! way of knowing what has happened to the record RMS has
    ! no way of knowing whether it has not been deleted and
    ! can still be access through its RFA, or whether it has
    ! been deleted and can not. In this case, RMS must search
    ! for the record by its RFA address, saved as part of the
    ! next record positioning context information, to confirm
    ! its status.
    IF .IRAB[IRB$L_CUR_VBN] NEQ .IRAB[IRB$L_POS_VBN]
    THEN
      FLAG[FND_BY_RRV] = 1
      ! If RMS has positioned to a RRV through the use of the
      ! current record information, and the VBN of that bucket is
      ! the same as the VBN of the current record's RFA address,
      ! then RMS has positioned to the RRV for the current
      ! record. Provided, the RRV is not marked deleted, RMS will
      ! be able to position to the current record by means of its
      ! RFA address.
    ELSE
      IF .STATUS
        AND
        .REC_ADDR[IRCSV_RRV]
        AND

```

```

: 1299 1359 5
: 1300 1360 5
: 1301 1361 5
: 1302 1362 5
: 1303 1363 5
: 1304 1364 5
: 1305 1365 5
: 1306 1366 5
: 1307 1367 5
: 1308 1368 5
: 1309 1369 5
: 1310 1370 5
: 1311 1371 5
: 1312 1372 5
: 1313 1373 5
: 1314 1374 5
: 1315 1375 5
: 1316 1376 5
: 1317 1377 5
: 1318 1378 5
: 1319 1379 5
: 1320 1380 5
: 1321 1381 5
: 1322 1382 5
: 1323 1383 5
: 1324 1384 5
: 1325 1385 5
: 1326 1386 5
: 1327 1387 5
: 1328 1388 5
: 1329 1389 5
: 1330 1390 5
: 1331 1391 5
: 1332 1392 5
: 1333 1393 5
: 1334 1394 5
: 1335 1395 5
: 1336 1396 5
: 1337 1397 4
: 1338 1398 4
: 1339 1399 4
: 1340 1400 4
: 1341 1401 4
: 1342 1402 4
: 1343 1403 4
: 1344 1404 4
: 1345 1405 4
: 1346 1406 4
: 1347 1407 4
: 1348 1408 4
: 1349 1409 4
: 1350 1410 4
: 1351 1411 5
: 1352 1412 5
: 1353 1413 5
: 1354 1414 5
: 1355 1415 5

```

```

NOT .REC_ADDR[IRCSV_DELETED]
THEN
    FLAG[FND_BY_RRV] = 1
    ! If the VBN of the current record is the same as the
    ! VBN of the current record's RFA address, and RMS
    ! was unable to position to a record (Prologue 3 - RRVs
    ! are deleted entirely although the record is just
    ! marked), or it was able to position to a deleted RRV
    ! (Prologue 1), then this indicates that the current
    ! record has been deleted. In such a situation a random
    ! search of the primary index will have to be done
    ! either to locate the next record itself (primary key
    ! duplicates NOT allowed), or to locate the current record.
ELSE
    FLAG[FND_BY_SRCH] = 1;
    ! It will be necessary to release the primary data bucket
    ! described by the current record information, if RMS has
    ! positioned to an RRV, and the bucket is still locked.
IF .STATUS
THEN
    RELEASE (IRAB[IRBSL_CURBDB]);
END
! RMS has successfully positioned to a non-RRV record by means
! of the next record positioning context information in the
! IRAB. This does not mean that RMS has positioned to the
! current record because it would be possible in a prologue 1 file
! for the current record to move, and its ID re-used by another
! record (ID recycling is allowed in one special circumstance).
! Therefore, that the record RMS has positioned to is in fact the
! current record must be verified by comparing the RRV in the
! record positioned to with the RFA address of the current record.
ELSE
    ! If the RRV of the record RMS has positioned to is not the
    ! same as the RFA address of the current record, then RMS
    ! has failed to position to the current record. If this is
    ! a prologue 3 file, then this represents a serious error
    ! because IDs within a bucket may not be recycled. If this
    ! is a prologue 1 file this does not represent a serious
    ! error because IDs maybe recycled; therefore, attempt to
    ! position to the current record by means of its RFA
    ! address.
IF NOT RMS$CHECK_RRV()
THEN
    BEGIN
        GLOBAL REGISTER
            COMMON_IO_STR;

```

:	1356	1416	5
:	1357	1417	5
:	1358	1418	5
:	1359	1419	5
:	1360	1420	6
:	1361	1421	5
:	1362	1422	5
:	1363	1423	5
:	1364	1424	5
:	1365	1425	5
:	1366	1426	5
:	1367	1427	5
:	1368	1428	5
:	1369	1429	5
:	1370	1430	5
:	1371	1431	5
:	1372	1432	5
:	1373	1433	4
:	1374	1434	4
:	1375	1435	4
:	1376	1436	3
:	1377	1437	3
:	1378	1438	3
:	1379	1439	3
:	1380	1440	3
:	1381	1441	3
:	1382	1442	3
:	1383	1443	3
:	1384	1444	3
:	1385	1445	3
:	1386	1446	4
:	1387	1447	4
:	1388	1448	4
:	1389	1449	4
:	1390	1450	4
:	1391	1451	4
:	1392	1452	4
:	1393	1453	4
:	1394	1454	4
:	1395	1455	4
:	1396	1456	4
:	1397	1457	4
:	1398	1458	4
:	1399	1459	4
:	1400	1460	4
:	1401	1461	4
:	1402	1462	4
:	1403	1463	4
:	1404	1464	4
:	1405	1465	4
:	1406	1466	4
:	1407	1467	4
:	1408	1468	4
:	1409	1469	4
:	1410	1470	4
:	1411	1471	4
:	1412	1472	4

P

```

RELEASE (IRAB[IRB$L_CURBDB]);

IF .IFAB[IFB$B_PLG_VER] GEQU PLG$C_VER_3
THEN
    RETURN RMSERR(BUG)
ELSE
    FLAG[FND_BY_RRV] = 1
END

! If the RRV of the record RMS has positioned to is the
! same as the RFA address of the current record, then RMS
! has successfully positioned to the current record.

! Look for the next non-deleted primary data record. Begin
! the search with the current record if and only if the
! state bit IRB$V_SKIP_NEXT is clear.
ELSE
    RETURN ON ERROR
    (RMS$FIND_NONDEL (NOT .IRAB[IRB$V_SKIP_NEXT]));
END;

! If RMS was unable to position to the current record by means
! of the current record information, then it may try to position
! to the current record by means of its RFA address. It will try
! to do so as long as it knows that the current record has not
! been deleted.
IF .FLAG[FND_BY_RRV]
THEN
    BEGIN
        LOCAL
            STATUS;

        ! If the file is Recovery Unit Journallable and write accessed,
        ! then make sure the primary data bucket containing the
        ! current primary data record is exclusively accessed.
        IF .IFAB[IFB$V_WRTACC]
            AND
            .IFAB[IFB$V_RU]
        THEN
            IRAB[IRB$B_CACHEFLGS] = CSH$M_LOCK;

        ! Position to the current record by means of its RFA address.
        STATUS = RMS$FIND_BY_RRV (.IRAB[IRB$L_POS_VBN],
            .IRAB[IRB$W_POS_ID],
            0);

        ! If RMS is unable to position to the current record by means
        ! of its RFA address, and the status is other than record
        ! deleted, then save the real reason for the failure in the
        ! RAB's SIV field and return an invalid internal condition.
        IF NOT .STATUS
    
```

```

: 1413
: 1414
: 1415
: 1416
: 1417
: 1418
: 1419
: 1420
: 1421
: 1422
: 1423
: 1424
: 1425
: 1426
: 1427
: 1428
: 1429
: 1430
: 1431
: 1432
: 1433
: 1434
: 1435
: 1436
: 1437
: 1438
: 1439
: 1440
: 1441
: 1442
: 1443
: 1444
: 1445
: 1446
: 1447
: 1448
: 1449
: 1450
: 1451
: 1452
: 1453
: 1454
: 1455
: 1456
: 1457
: 1458
: 1459
: 1460
: 1461
: 1462
: 1463
: 1464
: 1465
: 1466
: 1467
: 1468
: 1469

```

```

1473 4
1474 5
1475 4
1476 5
1477 5
1478 5
1479 5
1480 5
1481 5
1482 5
1483 5
1484 5
1485 5
1486 5
1487 4
1488 4
1489 4
1490 4
1491 4
1492 4
1493 4
1494 4
1495 4
1496 4
1497 4
1498 4
1499 4
1500 4
1501 4
1502 4
1503 4
1504 4
1505 4
1506 4
1507 4
1508 4
1509 4
1510 5
1511 5
1512 5
1513 5
1514 5
1515 5
1516 5
1517 6
1518 5
1519 5
1520 5
1521 5
1522 5
1523 5
1524 5
1525 5
1526 5
1527 5
1528 5
1529 4

```

```

AND
(.STATUS<0,16> NEQ RMSERR(DEL))
THEN
BEGIN
RAB[RAB$L_STV] = .STATUS;
RETURN RMSERR(BUG);
END

! If RMS is unable to position to the current record because
! it has been deleted, then indicate that a random search of
! the primary index to either locate the current record or
! to locate the next primary data record (primary key
! duplicates not allowed) must be done.
ELSE
IF NOT .STATUS
THEN
FLAG[FND_BY_SRCH] = 1

! RMS has successfully positioned to a primary data record
! through the use of the RFA address of the current record
! saved as part of the IRAB's next record positioning
! context. Before continuing, verify that the record RMS
! has positioned to is in fact the current record.
ELSE
! If the RRV of the record RMS has positioned to is
! not the same as the RFA address of the current record
! then RMS has failed to position to the current record
! by means of the next record context information. This
! represents a serious error, regardless of the
! prologue. Release the primary data bucket and return
! a status of invalid internal condition or bug error.
IF NOT RMSCHECK_RRV()
THEN
BEGIN
GLOBAL REGISTER
COMMON_IO_STR;

RELEASE (IRAB[IRB$L_CURBDB]);

RETURN RMSERR(BUG)
END

! If the RRV of the record RMS has positioned to is the
! same as the RFA address of the current record, then
! RMS has successfully positioned to the current record
! by means of its RFA address.

! Look for the next non-deleted primary data record.
! Begin the search with the current record if and
! only if the state bit IRB$V_SKIP_NEXT is clear.
ELSE

```

1470	P 1530	4	RETURN ON ERROR
1471	1531	4	(RMSFIND_NONDEL (NOT .IRAB[IRBSV_SKIP_NEXT]));
1472	1532	3	END;
1473	1533	3	
1474	1534	3	! If the current record has been deleted then a random search
1475	1535	3	! of the primary index must be done as part of positioning to
1476	1536	3	! the primary data record to be returned.
1477	1537	3	
1478	1538	3	IF .FLAG[FND_BY_SRCH]
1479	1539	3	THEN
1480	1540	4	BEGIN
1481	1541	4	
1482	1542	4	! If duplicate keys are not allowed and the current record is
1483	1543	4	! deleted then it will have been completely deleted from a
1484	1544	4	! prologue 3 file, and only a RRV will be left in a prologue 2
1485	1545	4	! file. Therefore, a search for the first non-deleted primary
1486	1546	4	! data record with a key equal to or greater than the key of
1487	1547	4	! the current record will position RMS to the record it is to
1488	1548	4	! return.
1489	1549	4	
1490	1550	4	IF NOT .IDX_DFN[IDXSV_DUPKEYS]
1491	1551	4	THEN
1492	1552	5	BEGIN
1493	1553	5	
1494	1554	5	! Position to the next first record with a key value equal
1495	1555	5	! to or greater than the key of the current record.
1496	1556	5	
1497	1557	5	IRAB[IRBSV_SRCHGE] = 1;
1498	1558	5	RETURN_ON_ERROR (RMSSEARCH_TREE());
1499	1559	5	
1500	1560	5	! Return the first non-deleted primary record encountered
1501	1561	5	! beginning the search with the record RMS has positioned
1502	1562	5	! to.
1503	1563	5	
1504	1564	5	RETURN_ON_ERROR (RMSFIND_NONDEL(1));
1505	1565	5	END
1506	1566	5	
1507	1567	5	! If duplicate primary keys are allowed then the deleted
1508	1568	5	! current record must still be somewhere in the file although
1509	1569	5	! marked deleted. Position to it by searching all records of
1510	1570	5	! key value identical to that of the current record until it
1511	1571	5	! is located.
1512	1572	5	
1513	1573	4	ELSE
1514	1574	5	BEGIN
1515	1575	5	
1516	1576	5	! Search for the first record in the file with the same
1517	1577	5	! primary key value as the key of the current record. If
1518	1578	5	! none is found then this represents a serious problem
1519	1579	5	! because the current record itself must always be
1520	1580	5	! somewhere in the file even if it has been deleted.
1521	1581	5	
1522	1582	5	IF NOT RMSSEARCH_TREE()
1523	1583	5	THEN
1524	1584	6	RETURN RMSERR(BUG)
1525	1585	6	
1526	1586	6	! RMS has successfully positioned to a primary data record

```

: 1527 1587 6
: 1528 1588 6
: 1529 1589 6
: 1530 1590 6
: 1531 1591 6
: 1532 1592 6
: 1533 1593 6
: 1534 1594 6
: 1535 1595 5
: 1536 1596 6
: 1537 1597 6
: 1538 1598 6
: 1539 1599 6
: 1540 1600 6
: 1541 1601 6
: 1542 1602 6
: 1543 1603 6
: 1544 1604 6
: 1545 1605 6
: 1546 1606 6
: 1547 1607 6
: 1548 1608 6
: 1549 1609 6
: 1550 1610 6
: 1551 1611 6
: 1552 1612 6
: 1553 1613 6
: 1554 1614 6
: 1555 1615 6
: 1556 1616 6
: 1557 1617 6
: 1558 1618 6
: 1559 1619 7
: 1560 1620 7
: 1561 1621 7
: 1562 1622 7
: 1563 1623 7
: 1564 1624 7
: 1565 1625 7
: 1566 1626 7
: 1567 1627 7
: 1568 1628 7
: 1569 1629 7
: 1570 1630 7
: 1571 1631 6
: 1572 1632 6
: 1573 1633 6
: 1574 1634 6
: 1575 1635 6
: 1576 1636 6
: 1577 1637 6
: 1578 1638 6
: 1579 1639 6
: 1580 1640 6
: 1581 1641 6
: 1582 1642 6
: 1583 1643 6

```

```

: with the same key as that of the current record.
: Somewhere in this bucket (or in any of the following
: continuation buckets if duplicates are allowed) must be
: the (marked) deleted current record. Position to the
: current record. After positioning to the current record,
: the next non-deleted primary data record in the file is
: the record RMS wants to return.
ELSE
  BEGIN
    LOCAL
      STATUS;

    : Since the current record must be in the file even if
    : it is just marked deleted, RMS's search criteria is
    : to sequentially find all records with key value
    : equal to that of the current record until the current
    : record is positioned to.
    STATUS = 1;

    DO
      : The record RMS has currently positioned to is
      : the current record when its RRV matches the RFA
      : address of the current record. In such a case
      : find the next non-deleted primary data record in
      : the file.
      IF RMSCHECK_RRV()
      THEN
        BEGIN
          RETURN ON_ERROR (RMSFIND_NONDEL (0));
          EXITLOOP;
          END

      : If the record RMS has positioned to is not the
      : current record (its RRV does not match the RFA
      : of the current record), then position to the next
      : record. RMSSEARCH_TREE will then determine
      : whether the key of this next record is the same
      : as the key of the current record.
      ELSE
        RMSGETNEXT_REC()

        WHILE (STATUS = RMSSEARCH_TREE());

      : If RMS was unable to find the current record in the
      : file then this represents a serious error because
      : current records can never be completely deleted from
      : the file while they remain current records. Return
      : status indicative of an invalid internal condition.
      IF NOT .STATUS
      THEN

```

```

: 1584 1644 7 IF .STATUS<0,16> EQL RMSERR(RNF)
: 1585 1645 6 THEN
: 1586 1646 7 RETURN RMSERR(BUG)
: 1587 1647 6 ELSE
: 1588 1648 6 RETURN .STATUS;
: 1589 1649 5 END;
: 1590 1650 4 END;
: 1591 1651 3 END;
: 1592 1652 2 END;
: 1593 1653 2
: 1594 1654 2 ! If RMS is positioning by primary key of reference then all RMS has to do
: 1595 1655 2 is set up the next record list context for this record, and return
: 1596 1656 2 success.
: 1597 1657 2
: 1598 1658 2 IF .IDX_DFN[IDX$B_KEYREF] EQL 0
: 1599 1659 2 THEN
: 1600 1660 2 BEGIN
: 1601 1661 2 IRAB[IRB$W_RFA_ID] = IRC$ ID(REC ADDR);
: 1602 1662 2 IRAB[IRB$L_RFA_VBN] = .BBLOCK[IRAB[IRB$L_CURBDB], BDB$L_VBN];
: 1603 1663 2 BBLOCK[IRAB[IRB$L_CURBDB], BDB$V_PRM] = T;
: 1604 1664 2 RETURN RMSSUC()
: 1605 1665 2 END
: 1606 1666 2
: 1607 1667 2 ! If RMS is positioning by means of an alternate key, it still must
: 1608 1668 2 position to the array element pointing to the non-deleted primary data
: 1609 1669 2 record to be returned.
: 1610 1670 2
: 1611 1671 2 ELSE
: 1612 1672 2 BEGIN
: 1613 1673 2
: 1614 1674 2 LOCAL
: 1615 1675 2 STATUS;
: 1616 1676 2
: 1617 1677 2 STATUS = 1;
: 1618 1678 2
: 1619 1679 2 ! Search all SIDRs having the same key as the SIDR RMS has positioned
: 1620 1680 2 to until the primary data record to be returned has been found, or
: 1621 1681 2 until there are no more SIDRs with this key value.
: 1622 1682 2
: 1623 1683 2 WHILE .STATUS
: 1624 1684 2 DO
: 1625 1685 2 BEGIN
: 1626 1686 2 IRAB[IRB$V_SRCHGE] = 0;
: 1627 1687 2 IRAB[IRB$V_PRM] = 1;
: 1628 1688 2
: 1629 1689 2 ! If RMS has successfully positioned to a non-deleted primary data
: 1630 1690 2 record, return success
: 1631 1691 2
: 1632 1692 2 IF STATUS = RMSSEARCH_SIDR()
: 1633 1693 2 THEN
: 1634 1694 2 RETURN .STATUS;
: 1635 1695 2
: 1636 1696 2 ! If RMS was unable to find a non-deleted primary data record with
: 1637 1697 2 an alternate key with the same value as the SIDR RMS has
: 1638 1698 2 positioned to, then move onto the next SIDR with a key greater
: 1639 1699 2 than that of the current SIDR.
: 1640 1700 2

```

```

: 1641 1701 5      IF .STATUS<0, 16> EQL RMSERR(RNF)
: 1642 1702 4      THEN
: 1643 1703 5      BEGIN
: 1644 1704 5      IRAB[IRB$V SRCHGE] = 1;
: 1645 1705 5      STATUS = RMSSEARCH_TREE();
: 1646 1706 5
: 1647 1707 5      ! If RMS is unable to find a SIDR on a greater or equal search
: 1648 1708 5      ! then this can only be because it has reached the
: 1649 1709 5      ! end-of-file, and so return the appropriate status.
: 1650 1710 5
: 1651 1711 6      IF .STATUS<0, 16> EQL RMSERR(RNF)
: 1652 1712 5      THEN
: 1653 1713 6      STATUS = RMSERR(EOF)
: 1654 1714 6
: 1655 1715 6      ! Otherwise continue the search for a non-deleted primary data
: 1656 1716 6      ! record with the first array element of the next SIDR.
: 1657 1717 6
: 1658 1718 5      ELSE
: 1659 1719 5      IRAB[IRB$W_SAVE_POS] = 0;
: 1660 1720 4      END;
: 1661 1721 4      END;
: 1662 1722 3      RETURN .STATUS;
: 1663 1723 3
: 1664 1724 3
: 1665 1725 3
: 1666 1726 2
: 1667 1727 2
: 1668 1728 1      END;

```

! { end of routine }

			34	BB	0000	RM\$POS_SEQ::			
			41	A9	94	00002	PUSHR	#^M<R2,R4,R5>	: 0961
			42	A9	B4	00005	CLRB	65(IRAB)	: 1106
			00A8	C9	D5	00008	CLRW	66(IRAB)	: 1107
				04	12	0000C	TSTL	168(IRAB)	: 1113
			00AC	C9	D5	0000E	BNEQ	1\$	
				03	13	00012	TSTL	172(IRAB)	: 1115
				009A	31	00014	BEQL	2\$	
			42	A9	40	8F 88	BRW	15\$	
				0000G	30	0001C	BISB2	#64, 66(IRAB)	: 1119
				0A	50	E8 0001F	BSBW	RMSSEARCH_TREE	: 1120
		82B2		8F	50	B1 C0022	BLBS	STATUS, 4\$	
					2E	13 00027	CMPW	STATUS, #33458	
					0288	31 00029	BEQL	8\$	
				52	20	A9 9E 0002C	BRW	56\$	
				50		62 D0 00030	MOVAB	32(R9), R2	: 1135
				55	18	A0 D0 00033	MOVL	(R2), R0	
					21	A7 95 00037	MOVL	24(R0), BKT_ADDR	
						04 12 0003A	TSTB	33(IDX_DFN)	: 1143
		06		66		03 E0 0003C	BNEQ	6\$	
				0E	04	A5 B1 00040	BBS	#3, (REC_ADDR), 7\$: 1145
						59 1A 00044	CMPW	4(BKT_ADDR), #14	: 1147
				0F	0D	A5 E9 00046	BGTRU	13\$	
							BLBC	13(BKT_ADDR), 9\$: 1155

		54		50	D0	0004A		MOVL	R0, BDB	1158		
				62	D4	0004D		CLRL	(R2)			
				7E	D4	0004F		CLRL	-(SP)			
				0000G	30	00051		BSPW	RMSRLSBKT			
		5E		04	C0	00054		ADDL2	#4, SP			
				78	11	00057	8\$:	BRB	16\$	1159		
		51		A5	D0	00059	9\$:	MOVL	8(BKT_ADDR), VBN	1165		
		54		20	A9	0005D		MOVL	32(IRAB), BDB	1166		
				20	A9	00061		CLRL	32(IRAB)			
				7E	D4	00064		CLRL	-(SP)			
				0000G	30	00066		BSBW	RMSRLSBKT			
		5E		04	C0	00069		ADDL2	#4, SP			
05	00A0	06		AA	E9	0006C		BLBC	6(IFAB), 10\$	1173		
		CA		01	E0	00070		BBS	#1, 160(IFAB), 11\$	1175		
				21	A7	95	00076	10\$:	TSTB	33(IDX_DFN)	1177	
				04	13	00079		BEQL	12\$			
		40		A9	01	90	0007B	11\$:	MOVB	#1, 64(IRAB)	1179	
		50		A7	9A	0007F	12\$:	MOVZBL	23(IDX_DFN), R0	1181		
7E		50		09	78	00083		ASHL	#9, R0, -(SP)			
				51	DD	00087		PUSHL	VBN			
				0000G	30	00089		BSBW	RMSGETBKT			
		5E		08	C0	0008C		ADDL2	#8, SP			
		97		50	E9	0008F		BLBC	STATUS, 3\$			
		52		20	A9	9E	0G092	MOVAB	32(R9), R2	1183		
		62		54	D0	00096		MOVL	BDB, (R2)			
		56		0E	A5	9E	00099	MOVAB	14(R5), REC_ADDR	1184		
				91	11	0009D		BRB	5\$	1125		
		42		A9	40	8F	8A	0009F	13\$:	BICB2	#64, 66(IRAB)	1191
				21	A7	95	000A4	TSTB	33(IDX_DFN)	1199		
				03	12	000A7		BNEQ	14\$			
				0155	31	000A9		BRW	39\$			
				76	A9	B4	000AC	14\$:	CLRW	118(IRAB)	1209	
				38	11	000AF		BRB	21\$	1113		
				21	A7	95	000B1	15\$:	TSTB	33(IDX_DFN)	1219	
				36	13	000B4		BEQL	22\$			
		42		A9	10	88	000B6	BISB2	#16, 66(IRAB)	1226		
				FE8D	30	000BA		BSBW	RMSFIND SIDR	1234		
		51		50	D0	000BD		MOVL	R0, STATUS			
		1E		51	E8	000C0		BLBS	STATUS, 20\$			
		8262		8F	51	B1	000C3	CMPW	STATUS, #33378	1236		
				14	13	000C8		BEQL	19\$			
		82B2		8F	51	B1	000CA	CMPW	STATUS, #33458	1243		
				07	12	000CF		BNEQ	17\$			
		50		827A	8F	3C	000D1	16\$:	MOVZWL	#33402, R0	1247	
				03	11	000D6		BRB	18\$			
		50		51	D0	000D8	17\$:	MOVL	STATUS, R0			
				01D6	31	000DB	18\$:	BRW	56\$			
		08		51	E9	000DE	19\$:	BLBC	STATUS, 21\$	1253		
03	05	A9		03	F1	000E1	20\$:	BBC	#3, 5(IRAB), 21\$			
				76	A9	B6	000E6	INCW	118(IRAB)	1255		
				0162	31	000E9	21\$:	BRW	49\$	1219		
				55	94	000EC	22\$:	CLRB	FLAG	1281		
		0A		06	AA	E9	000EE	BLBC	6(IFAB), 23\$	1298		
04	00A0	CA		01	E1	000F2		BBC	#1, 160(IFAB), 23\$	1300		
	40	A9		01	90	000F8		MOVB	#1, 64(IRAB)	1302		
		5C		00B8	C9	3C	000FC	23\$:	MOVZWL	184(IRAB), AP	1304	
				00A8	C9	DD	00101	PUSHL	168(IRAB)	1305		

			0000G	30	00105	BSBW	RMSFIND_BY_RFA		
		5E	04	C0	00108	ADDL2	#4, SP		
		11	50	E8	0010B	BLBS	STATUS, 25\$		1310
	82B2	8F	50	B1	0010E	CMPW	STATUS, #33458		1312
			07	13	00113	BEQL	24\$		
	8262	8F	50	B1	00115	CMPW	STATUS, #33378		1314
			BF	12	0011A	BNEQ	18\$		
		04	50	E9	0011C	BLBC	STATUS, 26\$		1323
30		66	03	E1	0011F	BBC	#3, (REC_ADDR), 30\$		1325
	00AC	C9	00A8	C9	D1	CC123	26\$: Cmpl	168(IRAB), 172(IRAB)	1342
			0B	12	0012A	BNEQ	27\$		
		0D	50	E9	0012C	BLBC	STATUS, 28\$		1355
09		66	03	E1	0012F	BBC	#3, (REC_ADDR), 28\$		1357
05		66	02	E0	00133	BBS	#2, (REC_ADDR), 28\$		1359
		55	01	88	00137	BISB2	#1, FLAG		1361
			03	11	0013A	BRB	29\$		
		55	02	88	0013C	BISB2	#2, FLAG		1375
		47	50	E9	0013F	BLBC	STATUS, 32\$		1381
		54	20	A9	D0	00142	29\$: MOVL	32(IRAB), BDB	1383
			20	A9	D4	00146	CLRL	32(IRAB)	
			7E	D4	00149	CLRL	-(SP)		
			0000G	30	0014B	BSBW	RMSRLSBK I		
		5E	04	C0	0014E	ADDL2	#4, SP		
			36	11	00151	BRB	32\$		1323
			FCCB	30	00153	30\$: BSBW	RMSCHECK_RRV		1409
		1B	50	E8	00156	BLBS	R0, 31\$		
		54	20	A9	D0	00159	MOVL	32(IRAB), BDB	1416
			20	A9	D4	0015D	CLRL	32(IRAB)	
			7E	D4	00160	CLRL	-(SP)		
			0000G	30	00162	BSBW	RMSRLSBKT		
		5E	04	C0	00165	ADDL2	#4, SP		
		03	00B7	CA	91	00168	CMPB	183(IFAB), #3	1418
			69	1E	0016D	BGEQU	36\$		
		55	01	88	0016F	BISB2	#1, FLAG		1422
			15	11	00172	BRB	32\$		1418
7E	05	A9	01	03	EF	00174	31\$: EXTZV	#3, #1, 5(IRAB), -(SP)	1435
		6E	6E	D2	0017A	MCOML	(SP), (SP)		
			FCC6	30	0017D	BSBW	RMSFIND_NONDEL		
		5E	04	C0	00180	ADDL2	#4, SP		
		03	50	E8	00183	BLBS	STATUS, 32\$		
			0083	31	00186	BRW	40\$		
		60	55	E9	00189	BLBC	FLAG, 38\$		1444
		0A	06	AA	E9	0018C	BLBC	6(IFAB), 33\$	1455
04	00A0	CA	01	E1	00190	BBC	#1, 160(IFAB), 33\$		1457
	40	A9	01	90	00196	MOVB	#1, 64(IRAB)		1459
			7E	D4	0C19A	33\$: CLRL	-(SP)		1463
		7E	00BA	C9	3C	0019C	MOVZWL	186(IRAB), -(SP)	1464
			00AC	C9	DD	001A1	PUSHL	172(IRAB)	1463
			0000G	30	001A5	BSBW	RMSFIND_BY_RRV		
		5E	0C	C0	001A8	ADDL2	#12, SP		
		15	50	E8	001AB	BLBS	STATUS, 35\$		1472
	8262	8F	50	B1	001AE	CMPW	STATUS, #33378		1474
			06	13	001B3	BEQL	34\$		
		0C	50	D0	001B5	MOVL	STATUS, 12(RAB)		1477
			1D	11	001B9	BRB	36\$		1478
		05	50	E8	001BB	34\$: BLBS	STATUS, 35\$		1488
		55	02	88	001BE	BISB2	#2, FLAG		1490

0A	50	20	A9	D0	00271	MOVL	32(IRAB), R0	:	1663
	A0		08	88	00275	BISB2	#8, 10(R0)	:	
	50		01	D0	00279	MOVL	#1, R0	:	1672
			36	11	0027C	BRB	56\$:	
	50		01	D0	0027E	MOVL	#1, STATUS	:	1677
	30		50	E9	00281	BLBC	STATUS, 56\$:	1683
42	A9		10	8A	00284	BICB2	#16, 66(IRAB)	:	1686
42	A9	80	8F	88	00288	BISB2	#28, 66(IRAB)	:	1687
			0000G	30	0028D	BSBW	RMSSEARCH_SIDR	:	1692
	21		50	E8	00290	BLBS	STATUS, 56\$:	
82B2	8F		50	B1	00293	CMPW	STATUS, #33458	:	1701
			E7	12	00298	BNEQ	54\$:	
	42		10	88	0029A	BISB2	#16, 66(IRAB)	:	1704
			0000G	30	0029E	BSBW	RMSSEARCH_TREE	:	1705
82B2	8F		50	B1	002A1	CMPW	STATUS, #33458	:	1711
			07	12	002A6	BNEQ	55\$:	
	50	827A	8F	3C	002A8	MOVZWL	#33402, STATUS	:	1713
			D2	11	002AD	BRB	54\$:	
		76	A9	B4	002AF	CLRW	118(IRAB)	:	1719
			CD	11	002B2	BRB	54\$:	1701
			34	BA	002B4	POPR	#^M<R2,R4,R5>	:	1728
			05	002B6	RSB			:	

: Routine Size: 695 bytes, Routine Base: RMSRMS3 + 01DF

```
: 1669      1729  1
: 1670      1730  1 END
: 1671      1731  1
: 1672      1732  0 ELUDOM
```

PSECT SUMMARY

Name	Bytes	Attributes
RMSRMS3	1174	NOVEC,NOWRT, RD, EXE,NOSHR, GBL, REL, CON, PIC,ALIGN(2)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[RMS.OBJ]RMS.L32;1	3109	86	2	154	00:00.4

```
: Information: 2
: Warnings: 0
```

RM3POSSEQ
V04-000

RM\$POS_SEQ

C 1
16-Sep-1984 01:56:09
14-Sep-1984 13:01:34

VAX-11 Bliss-32 V4.0-742
[RMS.SRC]RM3POSSEQ.B32;1

Page 40
(5)

RM3
V04

: Errors: 0

COMMAND QUALIFIERS

: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:RM3POSSEQ/OBJ=OBJ\$:RM3POSSEQ MSRC\$:RM3POSSEQ/UPDATE=(ENH\$:RM3POSSEQ)

: Size: 1174 code + 0 data bytes
: Run Time: 00:33.5
: Elapsed Time: 01:04.6
: Lines/CPU Min: 3104
: Lexemes/CPU-Min: 15804
: Memory Used: 274 pages
: Compilation Complete

