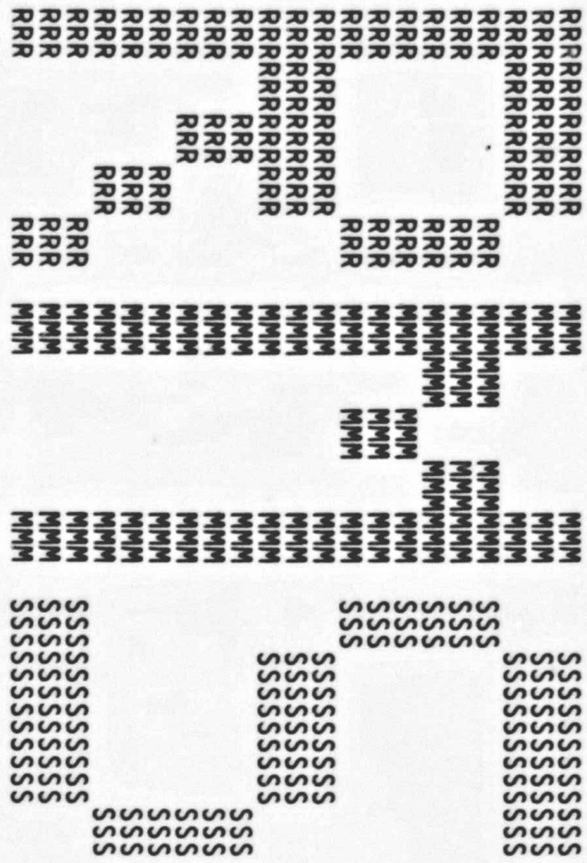


```
$2
Sy
---
NT9
NT9
NT9
```

```
RRRRRRRR    MM      MM    333333      GGGGGGGG  EEEEEEEEEE  TTTTTTTTTT
RRRRRRRR    MM      MM    333333      GGGGGGGG  EEEEEEEEEE  TTTTTTTTTT
RR      RR  MMMM  MMMM  33      33  GG          EE              TT
RR      RR  MMMM  MMMM  33      33  GG          EE              TT
RR      RR  MM  MM  MM          33  GG          EE              TT
RR      RR  MM  MM  MM          33  GG          EE              TT
RRRRRRRR    MM      MM        33    GG          EEEEEEEE        TT
RRRRRRRR    MM      MM        33    GG          EEEEEEEE        TT
RR  RR      MM      MM          33  GG  GGGGGG  EE              TT
RR  RR      MM      MM          33  GG  GGGGGG  EE              TT
RR    RR    MM      MM  33      33  GG      GG  EE              TT          ....
RR    RR    MM      MM  33      33  GG      GG  EE              TT          ....
RR      RR  MM      MM    333333      GGGGGG    EEEEEEEEEE      TT          ....
RR      RR  MM      MM    333333      GGGGGG    EEEEEEEEEE      TT          ....

LL            IIIIII      SSSSSSSS
LL            IIIIII      SSSSSSSS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II      SS
LL              II        SSSSSS
LL              II        SSSSSS
LL              II              SS
LL              II              SS
LL              II              SS
LL              II              SS
LLLLLLLLLL    IIIIII    SSSSSSSS
LLLLLLLLLL    IIIIII    SSSSSSSS
```

```
0 MODULE RM3GET (LANGUAGE (BLISS32) ,
0                 IDENT = 'V04-000'
0                 ) =
1 BEGIN
1 !
1 !*****************************************************************************
1 !*                                                                           *
1 !*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                  *
1 !*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                   *
1 !*  ALL RIGHTS RESERVED.                                                     *
1 !*                                                                           *
1 !*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED    *
1 !*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE    *
1 !*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER    *
1 !*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY    *
1 !*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY    *
1 !*  TRANSFERRED.                                                             *
1 !*                                                                           *
1 !*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE    *
1 !*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT    *
1 !*  CORPORATION.                                                             *
1 !*                                                                           *
1 !*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS    *
1 !*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                  *
1 !*                                                                           *
1 !*                                                                           *
1 !*****************************************************************************
1
1 !++
1 !
1 ! FACILITY:     RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
1 !
1 ! Abstract:
1 !                 This module implements the get and find record operations
1 !                 for the indexed file organization.
1 !
1 !
1 ! ENVIRONMENT:
1 !
1 !               VAX/VMS OPERATING SYSTEM
1 !
1 !--
1
1 !
1 ! AUTHOR:        E. H. MARISON     CREATION DATE:                18-APR-78  13:11
1 !
1 !
1 ! MODIFIED BY:
1 !
1 !       V03-025 JWT0193         Jim Teague                13-Aug-1984
1 !               Fix bug in re-accessing records after they have been
1 !               found to be locked.  If a process had done a $GET on
1 !               a record, then a $RELEASE, and then had to wait to
1 !               $GET the record a second time, too much context was
1 !               still around from the first $GET.  This caused problems
1 !               when the sought-after record had been deleted.  RMS
1 !               treated the $GET + $RELEASE + $GET case just like
```

```
;  58          0058  1 !                a $FIND + $GET case, and would end up with the wrong
;  59          0059  1 !                record.
;  60          0060  1 !
;  61          0061  1 !                Also, improve $GET/$FIND performance. Leave the
;  62          0062  1 !                infinite GET_RECORD loop immediately if GET_RECORD
;  63          0063  1 !                returns an unqualified success status.  Formerly,
;  64          0064  1 !                RMS was forced to grind through an unbelievably
;  65          0065  1 !                perverted IF test EVERY TIME it returned from
;  66          0066  1 !                GET_RECORD.
;  67          0067  1 !
;  68          0068  1 !        V03-024 TSK0001         Tamar Krichevsky        15-Jun-1983
;  69          0069  1 !                Change addressing mode to long relative for RM$RU_RECLAIM.
;  70          0070  1 !
;  71          0071  1 !        V03-023 MCN0015         Maria del C. Nasr       24-Mar-1983
;  72          0072  1 !                More linkages reorganization.
;  73          0073  1 !
;  74          0074  1 !        V03-022 TMK0015         Todd M. Katz            11-Mar-1983
;  75          0075  1 !                If RMS had to wait for a record lock, and it must re-position
;  76          0076  1 !                to the primary data record by calling RM$FIND_BY_RRV, then make
;  77          0077  1 !                sure the primary data bucket containing the record is locked
;  78          0078  1 !                exclusively if the possibility exists that some reclamation
;  79          0079  1 !                maybe done (the file is write accessed and RU Journallable).
;  80          0080  1 !
;  81          0081  1 !        V03-021 MCN0014         Maria del C. Nasr       24-Feb-1983
;  82          0082  1 !                Reorganize linkages
;  83          0083  1 !
;  84          0084  1 !        V03-020 TMK0014         Todd M. Katz            14-Jan-1983
;  85          0085  1 !                Add support for Recovery Unit Journalling and RU ROLLBACK
;  86          0086  1 !                Recovery of ISAM files. Support involves modifications to
;  87          0087  1 !                RM$GET3B and RM$GET_RECORD.
;  88          0088  1 !
;  89          0089  1 !                The purpose of the routines within this module is to retrieve
;  90          0090  1 !                a non-deleted primary data record by the user specified access
;  91          0091  1 !                mode. If during its search for such a record RMS in its
;  92          0092  1 !                low-level routines encounters records that are marked
;  93          0093  1 !                RU_DELETE, RMS will try and delete them for good at this time
;  94          0094  1 !                provided it has write access to the file and the Recovery Unit
;  95          0095  1 !                in which they were deleted has completed successfully.
;  96          0096  1 !
;  97          0097  1 !                If RMS is able to delete a primary data record marked RU_DELETE
;  98          0098  1 !                in these low-level routines, then RMS proceeds to continue
;  99          0099  1 !                looking for a non-deleted primary data record just as if it had
; 100          0100  1 !                encountered a deleted record in the first place. Likewise, if
; 101          0101  1 !                RMS is unable to delete a record that is marked RU_DELETE
; 102          0102  1 !                because it does not have write access to the file, it merely
; 103          0103  1 !                continues its search. However, if RMS is unable to delete the
; 104          0104  1 !                record for good in these low-level routines because the
; 105          0105  1 !                Recovery Unit in which it was marked RU_DELETE has not
; 106          0106  1 !                successfully terminated, then RMS returns this record as if it
; 107          0107  1 !                was the non-deleted primary data record to be returned, and
; 108          0108  1 !                lets a higher-level routine decide whether or not to wait for
; 109          0109  1 !                the Recovery Unit in which the record was deleted to complete,
; 110          0110  1 !                or to return an error to the user.
; 111          0111  1 !
; 112          0112  1 !                The routines within this module are the high-level routines
; 113          0113  1 !                which decide what to do with RU_DELETEd records that are
; 114          0114  1 !                returned from the low-level positioning routines.
```

```
;  115          0115  1 !
;  116          0116  1 !                1. If RMS is unable to lock such a record because another
;  117          0117  1 !                   process currently has it locked, then an RLK error is
;  118          0118  1 !                   returned.
;  119          0119  1 !
;  120          0120  1 !                2. If RMS is able to lock such a record, regardless of whether
;  121          0121  1 !                   it had to wait for it or not, then if it finds that the
;  122          0122  1 !                   record is not marked RU_DELETE it will return it provided
;  123          0123  1 !                   all other normal conditions have been met.
;  124          0124  1 !
;  125          0125  1 !                3. If on the other hand, RMS finds that the record is still
;  126          0126  1 !                   marked RU_DELETE after it has locked it, then it will delete
;  127          0127  1 !                   the record for good at this time (if the stream has write
;  128          0128  1 !                   access to the file), and continue the search for a
;  129          0129  1 !                   non-deleted primary data record provided the access mode is
;  130          0130  1 !                   not by RFA.
;  131          0131  1 !
;  132          0132  1 !                I have also made two other changes in support of RU Journalling
;  133          0133  1 !                and Recovery. First, the ROP bit RAB$V_NLK is totally ignored
;  134          0134  1 !                whenever a stream is currently within a Recovery Unit. Finally,
;  135          0135  1 !                it is also possible that a RU_UPDATE marked record might be
;  136          0136  1 !                re-formatted before releasing the bucket in which it is found
;  137          0137  1 !                provided the stream has write access to the file. The record
;  138          0138  1 !                being re-formatted in this case can only be the record that is
;  139          0139  1 !                to be returned as the non-deleted primary data record.
;  140          0140  1 !
;  141          0141  1 !                I have made an additional change to RM$GET_RECORD. If RMS is
;  142          0142  1 !                currently randomly positioning by key to what it thinks is the
;  143          0143  1 !                current record, then it query locks the current record to make
;  144          0144  1 !                sure that this record is in fact locked to avoid a window in
;  145          0145  1 !                which the record is deleted between the time the record lock is
;  146          0146  1 !                released, and the bucket in which the record is found is
;  147          0147  1 !                accessed. If the user has specified record waiting it is
;  148          0148  1 !                disabled for this query lock. Currently it is disabled by
;  149          0149  1 !                clearing the RAB$V_WAT bit if it is set, and then
;  150          0150  1 !                re-establishing its state after the query lock. The state bit
;  151          0151  1 !                IRB$V_NO_Q_WAIT maybe set to accomplish this same thing and it
;  152          0152  1 !                avoids modifying the user's contol block.
;  153          0153  1 !
;  154          0154  1 !                I have created a routine RM$POS_RFA whose functionality
;  155          0155  1 !                parallels that of RM$POS_SEQ and RM$POS_KEY. That is, the
;  156          0156  1 !                routine RM$GET_RECORD will call this routine whenever it is to
;  157          0157  1 !                position to the next primary data record by RFA instead of
;  158          0158  1 !                performing the positioning itself.
;  159          0159  1 !
;  160          0160  1 !        V03-019 TMK0013          Todd M. Katz            09-Nov-1982
;  161          0161  1 !                Fix a bug in record unlocking. Whenever RMS must wait for a
;  162          0162  1 !                record lock (the RAB$V_WAT ROP bit is set), and upon being
;  163          0163  1 !                granted the lock finds that the record it has been waited on has
;  164          0164  1 !                been deleted, RMS must perform a re-positioning. (There is one
;  165          0165  1 !                exception to this rule. If RMS was accessing the record by its
;  166          0166  1 !                RFA then the record deleted error is returned.) RMS must also
;  167          0167  1 !                perform a re-positioning whenever it is positioning by means of
;  168          0168  1 !                an alternate key and has had to wait for a record lock. As part
;  169          0169  1 !                of this re-positioning, RMS must release the lock it obtained
;  170          0170  1 !                during the prior positioning attempt. The problem is that RMS
;  171          0171  1 !                was using the wrong RFA when it went to release the record
```

```
;  172          0172  1 !                whenever it was re-positioning because the record it had to wait
;  173          0173  1 !                for had been deleted while it was waiting for it. The RFA it was
;  174          0174  1 !                using was the RFA of the current record. This record had been
;  175          0175  1 !                locked during the previous positioning operation, and had been
;  176          0176  1 !                unlocked during the first positioning attempt of the current
;  177          0177  1 !                operation. The lock RMS wants to release is for the record
;  178          0178  1 !                locked during the previous positioning attempt.
;  179          0179  1 !
;  180          0180  1 !                The fix for this problem is relatively straightforward. RMS
;  181          0181  1 !                never has to re-position unless it has had to wait for a record
;  182          0182  1 !                lock. Therefore, what I did was set the state bit OK_WAT_STATUS
;  183          0183  1 !                whenever a re-positioning has to be done instead of just
;  184          0184  1 !                setting it whenever the re-positioning is being done because
;  185          0185  1 !                RMS had to wait for a record lock while positioning along an
;  186          0186  1 !                alternate index. The setting of this state bit forces RMS to
;  187          0187  1 !                unlock the correct record during the re-positioning attempt.
;  188          0188  1 !
;  189          0189  1 !        V03-018 TMK0012         Todd M. Katz            29-Oct-1982
;  190          0190  1 !                Make sure that RMS has the index descriptor for the primary key
;  191          0191  1 !                before the size of the primary data record to be returned is
;  192          0192  1 !                determined, the record unpacked (if the file is a prologue 3
;  193          0193  1 !                file), and the record moved into the user's buffer.
;  194          0194  1 !
;  195          0195  1 !        V03-017 TMK0011         Todd M. Katz            11-Oct-1982
;  196          0196  1 !                Fix a record locking bug. Whenever the ROP bit RAB$V_WAT is set
;  197          0197  1 !                the possibility exists that RMS might have to wait for a record
;  198          0198  1 !                lock. If RMS is positioning by means of an alternate index, and
;  199          0199  1 !                has to wait for such a record lock, then it had to give up the
;  200          0200  1 !                SIDR bucket while it was waiting. Because it gave up the SIDR
;  201          0201  1 !                bucket, the information which it has inorder to update the NRP
;  202          0202  1 !                list can no longer be considered valid. Since there is no way
;  203          0203  1 !                for RMS is to easily re-access the SIDR bucket, RMS must
;  204          0204  1 !                re-position to it by re-calling GET_RECORD. Part of this
;  205          0205  1 !                re-positioning includes unlocking the very same primary data
;  206          0206  1 !                record which it had to wait for a record lock on. Unfortunately,
;  207          0207  1 !                GET_RECORD uses the NRP information to unlock primary data
;  208          0208  1 !                records, and RMS of course, didn't get to the point where it
;  209          0209  1 !                updated the NRP! Therefore, RMS is either not unlocking any
;  210          0210  1 !                record, or it is unlocking the wrong record. Both cases
;  211          0211  1 !                represent errors.
;  212          0212  1 !
;  213          0213  1 !                To fix this what I have done is added an input parameter to
;  214          0214  1 !                GET_RECORD. If it is set, RMS is re-positioning because of the
;  215          0215  1 !                above mentioned problem, and uses the RFA internally saved
;  216          0216  1 !                from the prior positioning attempt to unlock the record;
;  217          0217  1 !                otherwise, the RFA from the current record saved as part of the
;  218          0218  1 !                NRP context is used. Furthermore, whenever RMS  does such a
;  219          0219  1 !                re-positioning, it now notes that it had to do so because of
;  220          0220  1 !                an OK_WAT success status positioning on an alternate key of
;  221          0221  1 !                reference. If it is successful at re-positioning, it sets
;  222          0222  1 !                the status to OK_WAT which represents the status that it would
;  223          0223  1 !                have returned if the re-positioning had not been necessary.
;  224          0224  1 !
;  225          0225  1 !                I have made two additional changes concerning when
;  226          0226  1 !                re-positioning is required. First, if RMS is positioning by key
;  227          0227  1 !                value, and after waiting for a record lock finds that the record
;  228          0228  1 !                it has been waited for has been deleted, then RMS will perform a
```

```
;  229          0229  1 !                re-positioning to the next record which matches the search key
;  230          0230  1 !                in keybuffer 2 according to the characteristics of the search.
;  231          0231  1 !                Formerly, RMS was just returning a record deleted error, but I
;  232          0232  1 !                believe the other approach has more merit. Second, whenever RMS
;  233          0233  1 !                is positioning by an alternate key of reference (sequentially or
;  234          0234  1 !                randonmly by key value), and must wait for a record lock, then
;  235          0235  1 !                RMS must re-position to re-establish the NRP information for the
;  236          0236  1 !                SIDR. Formerly, this re-positioning was not done if RMS was
;  237          0237  1 !                performing a random $FIND. However, since the stream which has
;  238          0238  1 !                the record locked can delete the SIDR array positioned to by the
;  239          0239  1 !                waiting stream without deleting the actual primary data record
;  240          0240  1 !                (by means of an $UPDATE), then as the record eventually returned
;  241          0241  1 !                would not have the "correct" alternate key if re-positioning
;  242          0242  1 !                were not done, I believe that this requires this re-positioning
;  243          0243  1 !                to take place, even though the NRP is not going to be updated
;  244          0244  1 !                by this particular operation.
;  245          0245  1 !
;  246          0246  1 !                Finally, the last thing I did was make some changes on how the
;  247          0247  1 !                record unlocking is done when buffer errors are encountered
;  248          0248  1 !                during a $GET/$FIND. At this point the record has already been
;  249          0249  1 !                locked, and must be unlocked before control returns to the
;  250          0250  1 !                user. The routine GET_RECORD returns information in AP to
;  251          0251  1 !                RM$GET3B as to whether any special action is required to unlock
;  252          0252  1 !                this record on buffer errors. Unfortunately, AP is used
;  253          0253  1 !                throughout the remainder of RM$GET3B as input to record
;  254          0254  1 !                unpacking and key extraction; thus, its contents should a buffer
;  255          0255  1 !                error be detected and the record need to be unlocked, are
;  256          0256  1 !                unreliable. To fix this problem, I now set a flag bit on return
;  257          0257  1 !                from GET_RECORD if in fact special action will be required to
;  258          0258  1 !                unlock the record on buffer errors, and reference this bit
;  259          0259  1 !                in that circumstance rather than the AP.
;  260          0260  1 !
;  261          0261  1 !        V03-016 TMK0010         Todd M. Katz            29-Sep-1982
;  262          0262  1 !                If a file is a prologue 3 file with alternate keys, and RMS
;  263          0263  1 !                is positioning by means of an alternate key of reference, then
;  264          0264  1 !                RMS was not unpacking the record before returning it to the user
;  265          0265  1 !                because it assummed that the record had been unpacked during the
;  266          0266  1 !                positioning and there was no need to unpack it a second time.
;  267          0267  1 !                However, while this is true, RMS at this point does not know
;  268          0268  1 !                the unpacked record's size. Thus, for the time being RMS must
;  269          0269  1 !                always unpack the record before moving it into the user's
;  270          0270  1 !                record buffer if the file is a prologue 3 file.
;  271          0271  1 !
;  272          0272  1 !        V03-015 TMK0009         Todd M. Katz            09-Sep-1982
;  273          0273  1 !                The field IRB$B_SRCHFLAGS is now a word in size. Change all
;  274          0274  1 !                references to it.
;  275          0275  1 !
;  276          0276  1 !                Whenever RMS is positioning by means of an alternate key of
;  277          0277  1 !                reference (IRB$B_RP_KREF > 0), then there is never a need in the
;  278          0278  1 !                local routine GET_RECORD to extract the alternate key of the
;  279          0279  1 !                record positioned to into keybuffer 2. This is because as part
;  280          0280  1 !                of positioning to the primary data record from the SIDR in the
;  281          0281  1 !                first place, the SIDR key has already been extracted into
;  282          0282  1 !                keybuffer 2.
;  283          0283  1 !
;  284          0284  1 !                Eliminate all references to the routine RM$KEY_TYPE_CONV, since
;  285          0285  1 !                this routine doesn't do anything anyway.
```

```
;  286          0286  1 !
;  287          0287  1 !                The only time it is necessary to check for a valid packed
;  288          0288  1 !                decimal key is when the key type is packed decimal. It is never
;  289          0289  1 !                necessary to check for a valid packed decimal type when there is
;  290          0290  1 !                more than one segment and the file is a prologue 3 file. The
;  291          0291  1 !                packed decimal verfication routine no longer requires
;  292          0292  1 !                parameters.
;  293          0293  1 !
;  294          0294  1 !        V03-014 KBT0294         Keith B. Thompson       23-Aug-1982
;  295          0295  1 !                Reorganize psects
;  296          0296  1 !
;  297          0297  1 !        V03-013 TMK0008         Todd M. Katz            10-Aug-1982
;  298          0298  1 !                At the present time, when the accessing of a record by RFA
;  299          0299  1 !                fails, the error returned by RM$FIND_BY_RRV is the error that
;  300          0300  1 !                gets reported to the user. Change this so that if this routine
;  301          0301  1 !                returns an error of RMS$_EOF (because the RFA VBN is greater
;  302          0302  1 !                than the VBN of any primary data bucket), this error gets mapped
;  303          0303  1 !                into an error of RMS$_RNF.
;  304          0304  1 !
;  305          0305  1 !        V03-012 MCN0013         Maria del C. Nasr       10-Aug-1982
;  306          0306  1 !                Check for less than 0 on call to RM$COMPARE_KEY so that
;  307          0307  1 !                LIM check is done correctly.  This is to fix bug introduced
;  308          0308  1 !                by MCN0012.
;  309          0309  1 !
;  310          0310  1 !        V03-011 TMK0007         Todd M. Katz            19-Jun-1982
;  311          0311  1 !                Implement the RMS cluster NRP solution. Basically this involves
;  312          0312  1 !                removal of the NRP cells from system space, and the maintenance
;  313          0313  1 !                of the next record positioning context locally within the IRAB.
;  314          0314  1 !                Changes required to the routines in this module are as follows:
;  315          0315  1 !
;  316          0316  1 !                1. The routine SETUP_NRP_DATA now sets up the current record
;  317          0317  1 !                   context in the process local IRAB instead of in the
;  318          0318  1 !                   system-wide NRP cell.
;  319          0319  1 !
;  320          0320  1 !                2. The IRAB variables IRB$L_NEXT_VBN and IRB$W_NEXT_ID are
;  321          0321  1 !                   used to temporarily hold the RFA address of the "next"
;  322          0322  1 !                   primary data record until the updating of the local NRP
;  323          0323  1 !                   context takes place. This is because nothing in the local
;  324          0324  1 !                   NRP context maybe modified, until everything is modified!
;  325          0325  1 !
;  326          0326  1 !                3. The local routines must also be modified both to make use
;  327          0327  1 !                   of the next record positioning context now saved within
;  328          0328  1 !                   the IRAB instead of within a systemwide NRP cell.
;  329          0329  1 !
;  330          0330  1 !                4. If RMS encounters the end-of-file set the IRB$V_EOF bit.
;  331          0331  1 !                   This bit is also cleared after successfully positioning
;  332          0332  1 !                   randomly by key value. The former function of this bit
;  333          0333  1 !                   has now been taken over by the new bit IRB$V_CON_EOF.
;  334          0334  1 !
;  335          0335  1 !                5. Special processing is required for $GETs following random
;  336          0336  1 !                   $FINDs. A random $FIND does not change the notion of what
;  337          0337  1 !                   the next record is although it does change the notion of
;  338          0338  1 !                   what the current record is! Example with the record sequence
;  339          0339  1 !                   0 A B  -   sequential $GET to A, random $FIND to 0, $DELETE
;  340          0340  1 !                   0, followed by a sequential $GET returns B, the next record.
;  341          0341  1 !                   The random $FIND changed the current record to 0, but did
;  342          0342  1 !                   not change the next record to 0! The RMS cluster solution
```

```
;  343          0343  1 !                for NRP positioning handles this by keeping the current
;  344          0344  1 !                primary data record's RFA and the RFA of the primary data
;  345          0345  1 !                record for NRP positioning in separate fields. Most
;  346          0346  1 !                operations set all NRP fields and as a result the RFA
;  347          0347  1 !                address of the current primary data record and the RFA
;  348          0348  1 !                address of the primary data record used for NRP positioning
;  349          0349  1 !                are the same. However, a random $FIND will set only the
;  350          0350  1 !                current primary data record's RFA field. If the random
;  351          0351  1 !                $FIND is immediately followed by a sequential $GET, then it
;  352          0352  1 !                is only at that moment that the local NRP context is setup
;  353          0353  1 !                to return the randomly found record as the next record.
;  354          0354  1 !
;  355          0355  1 !            Also, it is no longer necessary within GET_RECORD to loop
;  356          0356  1 !            on calls to RM$POS_SEQ or RM$POS_KEY when these routines return
;  357          0357  1 !            RLK errors. An RLK error could occur only when positioning on
;  358          0358  1 !            an alternate index and signalled that re-positioning should be
;  359          0359  1 !            forced. This re-positioning is now handled at a much lower
;  360          0360  1 !            level, and there is no longer any need to force it.
;  361          0361  1 !
;  362          0362  1 !            During the performance optimization of TMK0005 one incorrect
;  363          0363  1 !            assumption was made: that no deleted records were encountered
;  364          0364  1 !            between the last record retrieved, whose key is in keybuffer 1,
;  365          0365  1 !            and the new record that has just been retrieved. If this is
;  366          0366  1 !            true, the optimization holds, but if it is not, we can not use
;  367          0367  1 !            the key of the last retrieved record to uncompress the key of
;  368          0368  1 !            the new record, because the compression of the key of the new
;  369          0369  1 !            record is based upon the intervening deleted records, and not
;  370          0370  1 !            the key of the last record. In such a situation, the key of
;  371          0371  1 !            the new record must be extracted, and re-expanded in the old
;  372          0372  1 !            way performing a bucket scan if necessary.
;  373          0373  1 !
;  374          0374  1 !            Finally, it will no longer be necessary to unpack the primary
;  375          0375  1 !            data record when the file is a prologue 3 file, and RMS is
;  376          0376  1 !            currently positioning by an alternate key since the record
;  377          0377  1 !            will have been already unpacked and is within the internal
;  378          0378  1 !            record buffer.
;  379          0379  1 !
;  380          0380  1 !        V03-010 MCN0012         Maria del C. Nasr       29-Jun-1982
;  381          0381  1 !                Allow keys of different data types other than string.
;  382          0382  1 !                Change all CH$COMPARE calls to RM$COMPARE_KEY to compare
;  383          0383  1 !                keys taking into consideration the different data types.
;  384          0384  1 !
;  385          0385  1 !        V03-009 TMK0006         Todd M. Katz            26-May-1982
;  386          0386  1 !                I have changed how the ROP=LIM key comparison is performed.
;  387          0387  1 !                Formerly, the routine RM$COMPARE_REC was being called. It was
;  388          0388  1 !                being called because the (incorrect) assumption existed that
;  389          0389  1 !                the key of the next record might have to be extracted and
;  390          0390  1 !                re-expanded, if key compression was enabled, in order to make
;  391          0391  1 !                the comparison. As it turns out, at this point in the
;  392          0392  1 !                operation, RMS has already extracted (and re-expanded if
;  393          0393  1 !                necessary) the key of the next record into keybuffer 2. Thus,
;  394          0394  1 !                in order to make the comparison, only a CH$COMPARE between
;  395          0395  1 !                the key in the user's key buffer and the key in keybuffer 2
;  396          0396  1 !                need be made. Thus, this comparison has now been made prologue
;  397          0397  1 !                independent, it is a performance optimization for all prologue
;  398          0398  1 !                versions, and the performance realized for prologue 3 files
;  399          0399  1 !                is considerable because it eliminates the need for one more
```

```
;  400          0400  1 !                bucket scan which was unnecessarily being done to re-expand
;  401          0401  1 !                the key of the next record.
;  402          0402  1 !
;  403          0403  1 !        V03-008 TMK0005         Todd M. Katz            26-May-1982
;  404          0404  1 !                Performance enhancement. After successfully positioning to the
;  405          0405  1 !                next record, RMS extracts its key into keybuffer 2. If key
;  406          0406  1 !                compression is enabled this mandates another bucket pass to
;  407          0407  1 !                re-expand the key. However, if RMS is positioning sequentially,
;  408          0408  1 !                then it has the key of the previous record retrieved saved in
;  409          0409  1 !                keybuffer 1. RMS can use this key to supply any characters
;  410          0410  1 !                front compressed off the key of the current record instead of
;  411          0411  1 !                performing another bucket pass to expand the key.
;  412          0412  1 !
;  413          0413  1 !        V03-007 TMK0004         Todd M. Katz            24-May-1982
;  414          0414  1 !                Performance enhancement. When performing a $GET on a prologue 3
;  415          0415  1 !                file, the record found must be unpacked before it is returned.
;  416          0416  1 !                Part of this unpacking includes extraction of the primary key
;  417          0417  1 !                from its position in front of the data record, and its
;  418          0418  1 !                re-expansion if key compression is enabled. But if we are
;  419          0419  1 !                positioning by primary key of reference then there is really no
;  420          0420  1 !                need to extract and re-expand the primary key because RMS
;  421          0421  1 !                already has it in the proper form within keybuffer 2. To
;  422          0422  1 !                signal to the routine RM$UNPACK_REC, that there is no need to
;  423          0423  1 !                extract and re-expand the primary key of the found data record,
;  424          0424  1 !                but that it maybe found in keybuffer 2, we initialize AP to 2
;  425          0425  1 !                before calling the routine when the key of reference is the
;  426          0426  1 !                primary key.
;  427          0427  1 !
;  428          0428  1 !        V03-006 LJA0008         Laurie Anderson         08-Apr-1982
;  429          0429  1 !                Must check for allocation of IDX_DFN, before access fields in
;  430          0430  1 !                it.  The IDX_DFN will not be allocated if there is an error
;  431          0431  1 !                returned from RM$KEY_DESC which is called by GET_RECORD.
;  432          0432  1 !
;  433          0433  1 !        V03-005 TMK0003         Todd M. Katz            01-Apr-1982
;  434          0434  1 !                If record locking is enabled, in GET_RECORD we lock the record
;  435          0435  1 !                we have found. If we had to wait for this record, the status
;  436          0436  1 !                returned is an alternate success (OK_WAT). We should be setting
;  437          0437  1 !                the IRB$V_UNLOCK_RP bit so that whenever we have finished
;  438          0438  1 !                with this record RMS will know to release it, but because our
;  439          0439  1 !                status is OK_WAT and not success, the current flow of control
;  440          0440  1 !                forces a return before this bit can be set. Therefore, the
;  441          0441  1 !                possibility exists that once a process has waited for a record
;  442          0442  1 !                lock and successfully locks the record, it will not release the
;  443          0443  1 !                lock unless explicitely told to do so (such as by an explicit
;  444          0444  1 !                $RELEASE). To avoid this undesirable possibility, we will
;  445          0445  1 !                make sure that IRB$V_UNLOCK_RP will be set even when we had
;  446          0446  1 !                to wait for a record lock.
;  447          0447  1 !
;  448          0448  1 !        V03-004 TMK0002         Todd M. Katz            26-Mar-1982
;  449          0449  1 !                Under two different set of circumstances we will have to
;  450          0450  1 !                release the record lock obtained in GET_RECORD.
;  451          0451  1 !
;  452          0452  1 !                1. If we have decided to make another iterative call to
;  453          0453  1 !                   GET_RECORD and we have locked a record within the last
;  454          0454  1 !                   call, then we must release this lock before attempting
;  455          0455  1 !                   to locate the next record in the current call.
;  456          0456  1 !
```

```
0457  1 !                2. If we are currently performing a random $FIND/$GET, and
0458  1 !                   we must wait in our attempt to lock the record we have
0459  1 !                   found (RAB$V_WAT is set), and upon returning and reaccessing
0460  1 !                   the bucket we found that this record has been deleted by
0461  1 !                   the stream that previously had it locked, then we must
0462  1 !                   release our lock on this deleted record before returning
0463  1 !                   the status of deleted record from RM$GET3B.
0464  1 !
0465  1 !                Both of these record lock releases may be signaled by setting
0466  1 !                the IRAB bit IRB$L_UNLOCK_RP within GET_RECORD at the
0467  1 !                appropriate time. The record will then be locked either within
0468  1 !                GET_RECORD in the former case or within RM$GET3B in the latter.
0469  1 !
0470  1 !        V03-003 TMK0001         Todd M. Katz            24-Mar-1982
0471  1 !                Change all references to te keybuffers to use the macro
0472  1 !                KEYBUF_ADDR
0473  1 !
0474  1 !                If an error status of RLK is returned on an attempt to
0475  1 !                "get" a data record, try again until the record is retrieved
0476  1 !                or a different error is returned. This will only occur when
0477  1 !                our key of reference is other than key 0, and someone else had
0478  1 !                the primary data (or RRV) bucket locked when we attempted to
0479  1 !                access it from the SIDR. The SIDR bucket must be released
0480  1 !                and we have to reaccess it inorder to avoid a potential
0481  1 !                deadlock situation, and returning an error of RLK will now
0482  1 !                guarentee that this is what will happen.
0483  1 !
0484  1 !                If the attempt to sequentially access a record results in a
0485  1 !                status of record deleted being returned from GET_RECORD,
0486  1 !                attempt to sequentially retrieve the very next record, and
0487  1 !                continue doing this until some other status is returned. This
0488  1 !                situation can develop if we try for a record lock and end up
0489  1 !                waiting (the ROP WAT bit is set) for it. While we are waiting
0490  1 !                the process (or stream) which has the record lock deletes it.
0491  1 !                When control returns to this process, the status it gets back
0492  1 !                indicates that it had to wait, and so it reacesses the bucket
0493  1 !                the record was in (it had to release it when it went for the
0494  1 !                record lock) and now finds the record is deleted and returns
0495  1 !                that status.
0496  1 !
0497  1 !                When control returns to RM$GETRECORD from RM$POS_SEQ,
0498  1 !                RM$POS_KEY, or RM$FIND_BY_RRV with a success status, the next
0499  1 !                record has been found, the bucket containing it has been
0500  1 !                locked, and the IRAB fields IRB$L_RFA_VBN, IRB$W_RFA_ID, and
0501  1 !                IRB$W_SAVE_POS contain the information necessary to update the
0502  1 !                NRP context to that of the "found" record. If a decision is
0503  1 !                made to lock the record, and RMS has to stall for the record
0504  1 !                lock (RAB$V_WAT is set and some other stream has the record
0505  1 !                locked) then when the lock is obtained, the bucket containing
0506  1 !                the record is no longer locked (if we have to wait for the
0507  1 !                record lock we must release the lock on the bucket to avoid the
0508  1 !                possibility of deadlock), and the NRP updating information
0509  1 !                within the IRAB can no longer be considered valid because the
0510  1 !                bucket containing the record might have split moving the record
0511  1 !                to the new bucket, and the record itself might even have been
0512  1 !                deleted. If we are accessing this file by its primary key,
0513  1 !                then as its record pointer (RP) information is still valid,
```

```
;  514          0514  1 !                we can call RM$FIND_BY_RRV to lock the bucket. Once the bucket
;  515          0515  1 !                has been again locked, the information necessary to update
;  516          0516  1 !                the next record context can be obtained and we can proceed.
;  517          0517  1 !                However, if we are accessing this record by an alternate
;  518          0518  1 !                key and we stall, then there is no easy way to reobtain the
;  519          0519  1 !                next record context information necessary for later updating of
;  520          0520  1 !                the NRP list. Thus, if the primary data record which we have
;  521          0521  1 !                locked is not deleted, and if we are performing a sequential
;  522          0522  1 !                $FIND, or a $GET operation (for which NRP list updating is
;  523          0523  1 !                mandatory), then we have no choice but repeat this lengthy
;  524          0524  1 !                process from the beginning. This is done BYby noting on return
;  525          0525  1 !                from GET_RECORD, that we have an alternate success status
;  526          0526  1 !                (OK_WAT), that we our key of reference is not the primary key,
;  527          0527  1 !                and that we have not locked the primary data bucket.
;  528          0528  1 !
;  529          0529  1 !        V03-002 LJA0006         Laurie Anderson         23-Mar-1982
;  530          0530  1 !                If the get record caused an RTB error, then the primary key
;  531          0531  1 !                was not copied into an RMS internal buffer.  This key buffer
;  532          0532  1 !                is used to avoid un-locking a record during a random access
;  533          0533  1 !                for an exact match by key when that record is the current
;  534          0534  1 !                record.
;  535          0535  1 !
;  536          0536  1 !        V03-001 KPL0009         Peter Lieberwirth       17-Mar-1982
;  537          0537  1 !                Set UNLOCK_RP on errors reaccessing record after successful
;  538          0538  1 !                wait for record lock.  This will cause the record to be
;  539          0539  1 !                unlocked on the way out.
;  540          0540  1 !
;  541          0541  1 !                Add sutitles.
;  542          0542  1 !
;  543          0543  1 !        V02-025 DJD0001         Darrell Duffy           1-March-1982
;  544          0544  1 !                Clean up probing of input parameters
;  545          0545  1 !
;  546          0546  1 !        V02-024 KPL0008         Peter Lieberwirth        5-Nov-1981
;  547          0547  1 !                Add support for PUT to EOF by clearing internal EOF flag on
;  548          0548  1 !                random gets, and returning RMS$_EOF on sequential gets.
;  549          0549  1 !
;  550          0550  1 !        V02-023 KPL0007         Peter Lieberwirth        7-Oct-1981
;  551          0551  1 !                Fix bug on reacessing buffer logic on secondary key.
;  552          0552  1 !
;  553          0553  1 !        V02-022 KPL0006         Peter Lieberwirth        2-Oct-1981
;  554          0554  1 !                Fix bugs related to interaction of WAT bit set when QUERY_LCK
;  555          0555  1 !                is called.  When re-accessing same record, don't WAT even if
;  556          0556  1 !                user said to until real lock logic.  Also, when QUERY_LCK
;  557          0557  1 !                called with WAT set in lock logic of GET_RECORD, remember to
;  558          0558  1 !                reaccess the bucket if RMS stalled.  Fix reaccess logic to
;  559          0559  1 !                work with secondary keys.  (Oops!)
;  560          0560  1 !
;  561          0561  1 !        V02-021 KPL0005         Peter Lieberwirth       23-Aug-1981
;  562          0562  1 !                Fix incorrect and misleading commentary due to V02-018.
;  563          0563  1 !                Also, allocate a temporary variable more efficently.
;  564          0564  1 !
;  565          0565  1 !        V02-020 MCN0011         Maria del C. Nasr       24-Jul-1981
;  566          0566  1 !                Implement key type conversion.
;  567          0567  1 !
;  568          0568  1 !        V02-019 MCN0010         Maria del C. Nasr       23-Jul-1981
;  569          0569  1 !                Incorporate all the following changes:
;  570          0570  1 !
```

```
0571  1 !                Use RM$REC_OVHD, and user's buffer to get key
0572  1 !                Use key buffer 4 to unpack primary key.
0573  1 !                Include code for unpacking of prologue 3 data records.
0574  1 !                Change calling sequence of RM$FIND_BY_RRV.
0575  1 !                Increase size of record identifier to a word in the IRB,
0576  1 !                NRP, and RLB.
0577  1 !                Modify routine to handle new prologue 3 data structure
0578  1 !                changes (base level 1).
0579  1 !
0580  1 !        V02-018 KPL0004         P. Lieberwirth          15-Jan-1981   3:15
0581  1 !                Change GET_RECORD to reaccess bucket if it had to be given
0582  1 !                up for wait on record.  Implements new ROP functionality
0583  1 !                implied by WAT and REA.
0584  1 !
0585  1 !        V02-017 SPR33597        P. Lieberwirth          24-Nov-1980  10:00
0586  1 !                Fix bug where omitted fetch operator caused incorrect test
0587  1 !                for validity of NRP.  Bug caused incorrect operation on
0588  1 !                sequential $FINDs.  Clarify some commentary by cleaning up
0589  1 !                some spelling mistakes, and explaining FIND some more.
0590  1 !
0591  1 !        V02-016 REFORMAT        K. E. Kinnear           24-Jul-1980  9:54
0592  1 !
0593  1 !        V02-015 CDS0073         C. D. Saether           17-Jan-1980  2:35
0594  1 !                Restructure current record unlocking logic to add check
0595  1 !                when duplicates aren't allowed to avoid record lock window
0596  1 !                and reaccessing current record.
0597  1 !
0598  1 !        V02-014 PSK0005         P. S. Knibbe            18-Dec-1979  5:00
0599  1 !                Check that packed decimal keys are in the correct format.
0600  1 !
0601  1 ! REVISION HISTORY:
0602  1 !
0603  1 !        V02-013                 C. D. Saether           12-Jul-1979  11:30
0604  1 !                Level calling RM$COMPARE_REC should be -1.
0605  1 !
0606  1 !        V01-012                 W. Koenig               6-Feb-1979  17:19
0607  1 !                Fill in user's RFA after some other checks.
0608  1 !
0609  1 !        V01-011                 W. Koenig               6-Dec-1978  10:19
0610  1 !                Implement RMS$_OK_LIM.
0611  1 !
0612  1 !        V01-010                 W. Koenig               5-Dec-1978  10:25
0613  1 !                Don't return DCT field.
0614  1 !
0615  1 !        V01-009                 W. Koenig               24-Oct-1978  14:02
0616  1 !                Make changes caused by sharing conventions.
0617  1 !
0618  1 !        V01-008                 W. Koenig               5-Oct-1978  14:02
0619  1 !                Zero all the NRP flags when resetting the NRP data.
0620  1 !
0621  1 !        V01-007                 W. Koenig               26-Sep-1978  16:42
0622  1 !                Don't zero the RP information after a successful get or
0623  1 !                sequential find.
0624  1 !
0625  1 !        V01-006                 W. Koenig               26-Sep-1978  13:15
0626  1 !                Can no longer zero out RP_SECOND as a longword.
0627  1 !
```

```
0628  1 !       V01-005                 C. D. Saether           21-Sep-1978  16:44
0629  1 !               Clear SRCFLAGS always.
0630  1 !
0631  1 !       V01-004                 W. Koenig               21-Sep-1978  15:50
0632  1 !               Return the data to the user on any seccess, not just ''suc''.
0633  1 !
0634  1 !       V01-003                 C. D. Saether           20-Sep-1978  16:25
0635  1 !               Clear NRP update flags when storing NRP.
0636  1 !
0637  1 !       V01-002                 C. D. Saether           12-Sep-1978  15:21
0638  1 !               Remove NXTBDB setup on RFA access.
0639  1 !
0640  1 !*****
0641  1
0642  1 LIBRARY 'RMSLIB:RMS';
0643  1
0644  1 REQUIRE 'RMSSRC:RMSIDXDEF';
0709  1
0710  1 ! Define default psects for code
0711  1 !
0712  1 PSECT
0713  1     CODE = RM$RMS3(PSECT_ATTR),
0714  1     PLIT = RM$RMS3(PSECT_ATTR);
0715  1
0716  1 ! Linkages.
0717  1 !
0718  1 LINKAGE
0719  1     L_COMPARE_KEY,
0720  1     L_JSB,
0721  1     L_JSB01,
0722  1     L_PRESERVE1,
0723  1     L_QUERY_AND_LOCK,
0724  1     L_RABREG,
0725  1     L_RABREG_67,
0726  1     L_RABREG_7,
0727  1     L_REC_OVHD,
0728  1
0729  1     ! Local Linkages.
0730  1     !
0731  1     L_GET_RECORD = JSB () :
0732  1                    GLOBAL (COMMON_RABREG, R_REC_ADDR, R_IDX_DFN)
0733  1                    NOPRESERVE (2, 3, 4, 5),
0734  1     L_SETUP_NRP  = JSB () :
0735  1                    GLOBAL (COMMON_RABREG, R_IDX_DFN)
0736  1                    NOPRESERVE (2, 3, 4, 5);
0737  1
0738  1 ! Forward Routines.
0739  1 !
0740  1 FORWARD ROUTINE
0741  1     GET_RECORD          : L_GET_RECORD;
0742  1
0743  1 ! External Routines.
0744  1 !
0745  1 EXTERNAL ROUTINE
0746  1     RM$COMPARE_KEY      : RL$COMPARE_KEY,
0747  1     RM$FIND_BY_RRV      : RL$RABREG_67,
0748  1     RM$KEY_DESC         : RL$RABREG_7,
```

```
;  685          0749  1     RM$LOCK              : RL$QUERY_AND_LOCK,
;  686          0750  1     RM$NOREAD_LONG       : RL$JSB,
;  687          0751  1     RM$NOWRT_LONG        : RL$JSB,
;  688          0752  1     RM$PCKDEC_CHECK      : RL$RABREG_7,
;  689          0753  1     RM$POS_KEY           : RL$RABREG_67,
;  690          0754  1     RM$POS_RFA           : RL$RABREG_67,
;  691          0755  1     RM$POS_SEQ           : RL$RABREG_67,
;  692          0756  1     RM$QUERY_LCK         : RL$QUERY_AND_LOCK,
;  693          0757  1     RM$RECORD_ID         : RL$RABREG_67,
;  694          0758  1     RM$RECORD_KEY        : RL$PRESERVE1,
;  695          0759  1     RM$RECORD_VBN        : RL$PRESERVE1,
;  696          0760  1     RM$REC_OVHD          : RL$REC_OVHD,
;  697          0761  1     RM$RLSBKT            : RL$PRESERVE1,
;  698          0762  1     RM$RU_RECLAIM        : RL$RABREG_67 ADDRESSING_MODE( LONG_RELATIVE ),
;  699          0763  1     RM$UNLOCK            : RL$QUERY_AND_LOCK,
;  700          0764  1     RM$UNPACK_REC        : RL$JSB01;
;  701          0765  1
```

```
1 %SBTTL 'SETUP_NRP_DATA'
1 ROUTINE SETUP_NRP_DATA : L_SETUP_NRP NOVALUE =
1
1 !++
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       This routine saves the next record positioning data
1 !       in the IRAB from the temporary IRAB locations filled
1 !       in during the positioning to the primary data record.
1 !
1 ! CALLING SEQUENCE:
1 !
1 !       SETUP_NRP_DATA()
1 !
1 ! INPUT PARAMETERS:
1 !       NONE
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       IRAB
1 !           IRB$L_FIRST_ID  - Current SIDR's first SIDR array element ID
1 !           IRB$L_FIRST_VBN - Current SIDR's first SIDR array element VBN
1 !           IRB$L_KEYBUF    - Pointer to keybuffers (to access keybuffer 2)
1 !           IRB$W_NEXT_ID   - ID of current primary data record
1 !           IRB$L_NEXT_VBN  - VBN of current primary data record
1 !           IRB$W_RFA_ID    - ID of current record (SIDR/primary)
1 !           IRB$L_RFA_VBN   - VBN of current record (SIDR/primary)
1 !           IRB$B_RP_KREF   - Key of reference used to retrieve user data record
1 !           IRB$W_SAVE_POS  - Number of elements before current SIDR element
1 !       IFAB
1 !           IFB$W_KBUFSZ    - Size of keybuffer (to access keybuffer 2)
1 !
1 ! OUTPUT PARAMETERS:
1 !       NONE
1 !
1 ! IMPLICIT OUTPUTS:
1 !       IRAB
1 !           IRB$W_CUR_COUNT - Number of elements before current SIDR element
1 !           IRB$W_CUR_ID    - ID of current record (SIDR/primary)
1 !           IRB$B_CUR_KREF  - Key of reference of current record (SIDR/primary)
1 !           IRB$L_CUR_VBN   - VBN of current record (SIDR/primary)
1 !           IRB$V_EOF       - clear indicating stream is not at end-of-file
1 !           IRB$L_KEYBUF    - Pointer to keybuffers (to access keybuffer 1)
1 !           IRB$W_POS_ID    - ID of primary data record for NRP positioning
1 !           IRB$L_POS_VBN   - VBN of primary data record for NRP positioning
1 !           IRB$L_SIDR_VBN  - Current SIDR's first SIDR array element VBN
1 !           IRB$W_SIDR_ID   - Current SIDR's first SIDR array element ID
1 !           IRB$W_UDR_ID    - ID of current primary data record
1 !           IRB$L_UDR_VBN   - VBN of current primary data record
1 !
1 ! ROUTINE VALUE:
1 !       NONE
1 !
1 !--
1
2     BEGIN
```

```
;  760          0823  2
;  761          0824  2      EXTERNAL REGISTER
;  762          0825  2          R_IDX_DFN_STR,
;  763          0826  2          COMMON_RAB_STR;
;  764          0827  2
;  765          0828  2      ! Indicate that this stream is no longer at the file's end of file.
;  766          0829  2      !
;  767          0830  2      IRAB[IRB$V_EOF] = 0;
;  768          0831  2
;  769          0832  2      ! Move the VBN of the current record into the appropriate IRAB location
;  770          0833  2      !
;  771          0834  2      IRAB[IRB$L_CUR_VBN] = .IRAB[IRB$L_RFA_VBN];
;  772          0835  2
;  773          0836  2      ! If the current record happens to also be the primary data record, then
;  774          0837  2      ! move its ID into the appropriate IRAB location.
;  775          0838  2      !
;  776          0839  2      IF .IRAB[IRB$B_RP_KREF] EQLU 0
;  777          0840  2      THEN
;  778          0841  2          IRAB[IRB$W_CUR_ID] = .IRAB[IRB$W_RFA_ID]
;  779          0842  2
;  780          0843  2      ! If the current record happens to be a SIDR, then it has no ID to save,
;  781          0844  2      ! and instead save the SIDR first array element's VBN and ID (this uniquely
;  782          0845  2      ! indetifies the SIDR), and the number of array elements preceeding the
;  783          0846  2      ! current element (which points to the primary data record that is being
;  784          0847  2      ! retrieved).
;  785          0848  2      !
;  786          0849  2      ELSE
;  787          0850  3          BEGIN
;  788          0851  3          IRAB[IRB$W_CUR_COUNT] = .IRAB[IRB$W_SAVE_POS];
;  789          0852  3          IRAB[IRB$L_SIDR_VBN]  = .IRAB[IRB$L_FIRST_VBN];
;  790          0853  3          IRAB[IRB$W_SIDR_ID]   = .IRAB[IRB$W_FIRST_ID];
;  791          0854  2          END;
;  792          0855  2
;  793          0856  2      ! Move the RFA of the current primary data record from its temporary
;  794          0857  2      ! location into the local NRP context and make it both the current primary
;  795          0858  2      ! data record and the primary data record for NRP positioning.
;  796          0859  2      !
;  797          0860  2      IRAB[IRB$L_UDR_VBN] = .IRAB[IRB$L_NEXT_VBN];
;  798          0861  2      IRAB[IRB$W_UDR_ID]  = .IRAB[IRB$W_NEXT_ID];
;  799          0862  2
;  800          0863  2      IRAB[IRB$L_POS_VBN] = .IRAB[IRB$L_NEXT_VBN];
;  801          0864  2      IRAB[IRB$W_POS_ID]  = .IRAB[IRB$W_NEXT_ID];
;  802          0865  2
;  803          0866  2      ! Setup up the key of reference of the current record, and move the key of
;  804          0867  2      ! the current record into keybuffer 1.
;  805          0868  2      !
;  806          0869  2      IRAB[IRB$B_CUR_KREF] = .IRAB[IRB$B_RP_KREF];
;  807          0870  2      CH$MOVE(.IFAB[IFB$W_KBUFSZ], KEYBUF_ADDR(2), KEYBUF_ADDR(1));
;  808          0871  2      RETURN;
;  809          0872  2
;  810          0873  1      END;
```

```
.TITLE  RM3GET
.IDENT  \V04-000\

.EXTRN  RM$COMPARE_KEY, RM$FIND_BY_RRV
```

```
                                                .EXTRN  RM$KEY_DESC, RM$LOCK
                                                .EXTRN  RM$NOREAD_LONG, RM$NOWRT_LONG
                                                .EXTRN  RM$PCKDEC_CHECK
                                                .EXTRN  RM$POS_KEY, RM$POS_RFA
                                                .EXTRN  RM$POS_SEQ, RM$QUERY_LCK
                                                .EXTRN  RM$RECORD_ID, RM$RECORD_KEY
                                                .EXTRN  RM$RECORD_VBN, RM$REC_OVHD
                                                .EXTRN  RM$RLSBKT, RM$RU_RECLAIM
                                                .EXTRN  RM$UNLOCK, RM$UNPACK_REC

                                                .PSECT  RM$RMS3,NOWRT,  GBL,  PIC,2

        04   A9          02  8A 00000 SETUP_NRP_DATA:
                                        BICB2   #2, 4(IRAB)                          ; 0830
      00A8   C9      70   A9  D0 00004         MOVL    112(IRAB), 168(IRAB)          ; 0834
             00C2    C9  95 0000A         TSTB    194(IRAB)                          ; 0839
                     08  12 0000E         BNEQ    1$                                 ;
      00B8   C9      74   A9  B0 00010         MOVW    116(IRAB), 184(IRAB)          ; 0841
                     13  11 00016         BRB     2$                                 ;
      00C0   C9      76   A9  B0 00018 1$:     MOVW    118(IRAB), 192(IRAB)          ; 0851
      00B4   C9      7C   A9  D0 0001E         MOVL    124(IRAB), 180(IRAB)          ; 0852
      00BE   C9    0082   C9  B0 00024         MOVW    130(IRAB), 190(IRAB)          ; 0853
      00B0   C9      78   A9  D0 0002B 2$:     MOVL    120(IRAB), 176(IRAB)          ; 0860
      00BC   C9    0080   C9  B0 00031         MOVW    128(IRAB), 188(IRAB)          ; 0861
      00AC   C9      78   A9  D0 00038         MOVL    120(IRAB), 172(IRAB)          ; 0863
      00BA   C9    0080   C9  B0 0003E         MOVW    128(IRAB), 186(IRAB)          ; 0864
      00C3   C9    00C2   C9  90 00045         MOVB    194(IRAB), 195(IRAB)          ; 0869
             50    00B4   CA  3C 0004C         MOVZWL  180(IFAB), R0                 ; 0870
             50      60   A9  C0 00051         ADDL2   96(IRAB), R0                  ;
  60   B9    60    00B4   CA  28 00055         MOVC3   180(IFAB), (R0), @96(IRAB)    ;
                                05 0005C         RSB                                 ; 0873

; Routine Size:  93 bytes,    Routine Base:  RM$RMS3 + 0000

;  811          0874  1
```

```
%SBTTL 'RM$GET3B'
GLOBAL ROUTINE RM$GET3B : RL$RABREG =

!++
!
! FUNCTIONAL DESCRIPTION:
!
!       This routine implements the get/find operation for the
!       indexed file organization.
!
! CALLING SEQUENCE:
!
!       RM$GET3()
!
! INPUT PARAMETERS:
!
!       R11     Impure area pointer
!       R10     IFAB --  Pointer to IFAB
!       R9      IRAB -- Pointer to IRAB
!       R8      RAB -- pointer to users RAB
!               ROP field options (NLK,ULK,RLK,LOC,NXR)
!               RAC field = (SEQ, or KEY, or RFA)
!               RFA field if RAC = RFA
!               KBF,KSZ,KRF if RAC = KEY and KBF,KSZ if RAC = SEQ and LIM set
!               UBF,USZ -- if a GET
!
! IMPLICIT INPUTS:
!
!       IRAB fields:
!
!               IRB$V_UNLOCK_RP - current record should be unlocked before
!                                 accessing new record.
!               IRB$V_FIND_LAST - last operation was a FIND.
!               IRB$V_SKIP_NEXT - last operation was sequential, the record
!                                 described by the nrp info is to be skipped
!                                 and the record beyond it becomes the new
!                                 record.
!               IRB$L_KEYBUF      (key buffer 1 or 2, maybe 3)
!
!       IFAB fields:
!
!               IFB$B_PLG_VER
!               IFB$V_RU_RLK    - if set, perform pseudo record locking
!               IFB$V_RUP       - if set, Recovoery Unit is in progress
!               IFB$W_KBUFSZ    - size of each keybuffer
!               IFB$V_NORECLCK - record locking not required, i.e., not
!                                sharing the file and single stream only.
!               IFB$V_WRTACC -   if accessed for other than read only.
!
! OUTPUT PARAMETERS:
!
!       RAB fields:
!
!               RFA of record found
!               STV if io errors
!               RBF,RSZ -- if a GET
!
```

```
;  870          0932  1 ! IMPLICIT OUTPUTS:
;  871          0933  1 !
;  872          0934  1 !       IRAB fields:
;  873          0935  1 !
;  874          0936  1 !               IRB$V_UNLOCK_RP
;  875          0937  1 !               IRB$V_FIND_LAST
;  876          0938  1 !               IRB$V_SKIP_NEXT
;  877          0939  1 !               IRB$B_RP_KREF
;  878          0940  1 !               IRB$B_CUR_KREF
;  879          0941  1 !               IRB$L_KEYBUF     (key buffer 1 or 2, maybe 3)
;  880          0942  1 !               IRB$L_RBF        User buffer address and size
;  881          0943  1 !               IRB$W_RSZ
;  882          0944  1 !
;  883          0945  1 ! ROUTINE VALUE:
;  884          0946  1 !
;  885          0947  1 !       Internal RMS status code
;  886          0948  1 !
;  887          0949  1 ! SIDE EFFECTS:
;  888          0950  1 !
;  889          0951  1 !       Retrieved record maybe locked, and next record context is modified.
;  890          0952  1 !
;  891          0953  1 !--
;  892          0954  1
;  893          0955  2     BEGIN
;  894          0956  2
;  895          0957  2     BUILTIN
;  896          0958  2         AP,
;  897          0959  2         TESTBITSC;
;  898          0960  2
;  899          0961  2     EXTERNAL REGISTER
;  900          0962  2         COMMON_RAB_STR;
;  901          0963  2
;  902          0964  2     GLOBAL REGISTER
;  903          0965  2         R_REC_ADDR_STR,
;  904          0966  2         R_IDX_DFN_STR;
;  905          0967  2
;  906          0968  2     LOCAL
;  907          0969  2         FLAGS    : BLOCK[1],
;  908          0970  2         STATUS;
;  909          0971  2
;  910          0972  2     MACRO
;  911          0973  2         AP_STATUS        = 0,0,1,0 %,
;  912          0974  2         OK_WAT_STATUS    = 0,1,1,0 %,
;  913          0975  2         RU_DEL_STATUS    = 0,2,1,0 %;
;  914          0976  2
;  915          0977  2
;  916          0978  2     ! Continue to attempt to get the next record under the following
;  917          0979  2     ! circumstances:
;  918          0980  2     !
;  919          0981  2     ! 1. The status returned from GET_RECORD indicates the next record has
;  920          0982  2     !    been deleted and RMS's access mode is sequential or random by key
;  921          0983  2     !    value. Any key of reference. This can only happen if RMS has had
;  922          0984  2     !    to wait for a record lock to be granted (status returned will be
;  923          0985  2     !    RMS$_DEL), or RMS has positioned to and managed to lock a primary
;  924          0986  2     !    data record that is marked RU_DELETE (status returned will be 0).
;  925          0987  2     !
;  926          0988  2     ! 2. The status returned from GET_RECORD is an alternate success status
```

```
!     (OK_WAT), and the key of reference is NOT the primary key.
!
! 3. An IDX_DFN has been allocated for the key descriptor, indicating
!     there are no problems with the key of reference.

! Force the key descriptor to be initially zero.
!
IDX_DFN = 0;
FLAGS   = 0;

WHILE 1
DO
    BEGIN

    STATUS = GET_RECORD (.FLAGS[OK_WAT_STATUS]
                                    OR
                                TESTBITSC(FLAGS[RU_DEL_STATUS]));
    !
    ! If an unqualified success, avoid the contorted mass of
    ! logic below and exit immediately...
    !
    IF .STATUS<0,16> EQLU RMSSUC()
    THEN
        EXITLOOP;

    ! Check the key descriptor after return from GET_RECORD.
    ! If still zero, something wrong with the key of
    ! reference, so exit loop.
    !
    IF .IDX_DFN EQLU 0
    THEN
        EXITLOOP;

    IF NOT  ((.STATUS<0,16> EQLU RMSERR(DEL)
                    OR
             .STATUS EQLU 0)
                    AND
            .RAB[RAB$B_RAC] NEQU RAB$C_RFA)
    THEN
       IF NOT (.IDX_DFN[IDX$B_KEYREF] NEQU 0
                AND
             .STATUS<0,16> EQLU RMSSUC(OK_WAT))
        THEN
            EXITLOOP;

    ! Let us back off from the radical position above concerning the
    ! ambitious attempts to continue to get a record.  We should
    ! never go back for a record if all of the following are
    ! true:
    !       - status from GET_RECORD is RMS$_DEL
    !       - primary key of reference
    !       - random access (keyed OR  RFA access)
    !       - no dups on primary key
    !       - exact key match
    !
    ! Under these circumstances, it is at least useless to
    ! to back after a record, and sometimes downright WRONG!
```

```
!
!
IF (.STATUS<0,16> EQLU RMSERR(DEL)
        AND
    .IDX_DFN[IDX$B_KEYREF] EQLU 0
        AND
    .RAB[RAB$B_RAC] NEQU RAB$C_SEQ
        AND
     NOT .IDX_DFN[IDX$V_DUPKEYS]
        AND
     NOT (.RAB[RAB$V_KGE] OR .RAB[RAB$V_KGT]))
THEN
    BEGIN
    STATUS = RMSERR(RNF);
    EXITLOOP;
    END;

! If RMS has to perform a re-positioning, then either it had to have
! waited for a record lock, or it positioned to a RU_DELETE marked
! primary data record. Therefore, set the appropriate state bit either
! of which will cause the correct lock to be released during
! re-positioning, and so that the proper status will be returned if
! RMS is able to position to a record.
!
IF .STATUS NEQU 0
THEN
    FLAGS[OK_WAT_STATUS] = 1
ELSE
    FLAGS[RU_DEL_STATUS] = 1;

    CH$MOVE (.IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(1), KEYBUF_ADDR(2));
    END;

! NOTE: AP is 0 if no special action is needed to unlock the RP on errors
! produced due to user buffer/size errors. Otherwise it is 1. Save this
! status.
!
FLAGS[AP_STATUS] = .AP<0,1>;

! If RMS was successful at obtaining the next record, but at some earlier
! time was forced to do a re-positioning because it had to wait for a record
! lock, then change the status to an OK_WAT success. This can only happen
! when RMS is positioning by means of an alternate index.
!
IF  .FLAGS[OK_WAT_STATUS]
     AND
    .STATUS<0,16> EQLU RMSSUC()
THEN
    STATUS = RMSSUC(OK_WAT);

IRAB[IRB$V_FIND_LAST] = 0;

! Obtain user buffer address and size for later probe.
!
IRAB [IRB$L_RBF] = .RAB [RAB$L_RBF];
IRAB [IRB$W_RSZ] = .RAB [RAB$W_RSZ];
```

```
; 1041          1103  2     ! If the user has set the RAB$V_LIM bit in the ROP field on a sequential
; 1042          1104  2     ! $GET/$FIND RMS reports whether the specified key exceeds the key of the
; 1043          1105  2     ! record found.
; 1044          1106  2     !
; 1045          1107  2     IF .STATUS
; 1046          1108  2         AND
; 1047          1109  3         (.RAB[RAB$B_RAC] EQL RAB$C_SEQ)
; 1048          1110  2         AND
; 1049          1111  2         .RAB[RAB$V_LIM]
; 1050          1112  2     THEN
; 1051          1113  3         BEGIN
; 1052          1114  3
; 1053          1115  3         LOCAL
; 1054          1116  3             KBF_ADDR     : LONG,
; 1055          1117  3             KEYSIZE;
; 1056          1118  3
; 1057          1119  3         KEYSIZE = .RAB[RAB$B_KSZ];
; 1058          1120  3
; 1059          1121  3         IF .KEYSIZE EQL 0
; 1060          1122  3         THEN
; 1061          1123  3
; 1062          1124  3             IF .IDX_DFN[IDX$B_DATATYPE] EQL IDX$C_STRING
; 1063          1125  3               OR .IDX_DFN[IDX$B_SEGMENTS] GTR 1
; 1064          1126  3             THEN
; 1065          1127  4                 STATUS = RMSERR(KSZ)
; 1066          1128  3             ELSE
; 1067          1129  3                 KEYSIZE = .IDX_DFN[IDX$B_KEYSZ];
; 1068          1130  3
; 1069          1131  4         BEGIN
; 1070          1132  4
; 1071          1133  4         MAP
; 1072          1134  4             KEYSIZE      : BYTE;
; 1073          1135  4
; 1074          1136  4         IF .KEYSIZE GTRU .IDX_DFN[IDX$B_KEYSZ]
; 1075          1137  4         THEN
; 1076          1138  5             STATUS = RMSERR(KSZ)
; 1077          1139  3         END;
; 1078          1140  3
; 1079          1141  3         KBF_ADDR = .RAB [RAB$L_KBF];
; 1080        P 1142  3         IFNORD(KEYSIZE, .KBF_ADDR, IRAB[IRB$B_MODE],
; 1081          1143  3             STATUS = RMSERR(KBF));
; 1082          1144  3
; 1083          1145  3         IF .STATUS
; 1084          1146  3         THEN
; 1085          1147  4             BEGIN
; 1086          1148  4             AP = 3;                 ! Contiguous key compare
; 1087          1149  4
; 1088          1150  4             ! The key of the current record has been previously saved in
; 1089          1151  4             ! keybuffer 2, and may now be used to determine whether the user
; 1090          1152  4             ! specified key limit has been exceeded.
; 1091          1153  4             !
; 1092          1154  4             IF RM$COMPARE_KEY ( KEYBUF_ADDR(2), .KBF_ADDR, .KEYSIZE ) LSS 0
; 1093          1155  4             THEN
; 1094          1156  4                 STATUS = RMSSUC(OK_LIM);
; 1095          1157  3             END;
; 1096          1158  3
; 1097          1159  2         END;
```

```
; 1098          1160  2
; 1099          1161  2     IF .STATUS
; 1100          1162  2     THEN
; 1101          1163  3         BEGIN
; 1102          1164  3
; 1103          1165  3         IF .IRAB[IRB$V_FIND]
; 1104          1166  3         THEN
; 1105          1167  3
; 1106          1168  3             ! This is a find operation don't
; 1107          1169  3             ! return record etc.
; 1108          1170  3             !
; 1109          1171  4             BEGIN
; 1110          1172  4             IRAB[IRB$V_FIND_LAST] = 1;
; 1111          1173  4
; 1112          1174  4             ! Set up the next record context for non-random $FIND operations.
; 1113          1175  4             !
; 1114          1176  5             IF  (.RAB[RAB$B_RAC] EQL RAB$C_SEQ)
; 1115          1177  4             THEN
; 1116          1178  4                 SETUP_NRP_DATA()
; 1117          1179  4
; 1118          1180  4             ! If this is a random $FIND operation then save the RFA of the
; 1119          1181  4             ! found primary data record as the current primary data record.
; 1120          1182  4             !
; 1121          1183  4             ELSE
; 1122          1184  5                 BEGIN
; 1123          1185  5                 IRAB[IRB$L_UDR_VBN] = .IRAB[IRB$L_NEXT_VBN];
; 1124          1186  5                 IRAB[IRB$W_UDR_ID]  = .IRAB[IRB$W_NEXT_ID];
; 1125          1187  4                 END;
; 1126          1188  4
; 1127          1189  4             END
; 1128          1190  3         ELSE
; 1129          1191  3
; 1130          1192  3             ! Return the user the data on the record.
; 1131          1193  3             !
; 1132          1194  4             BEGIN
; 1133          1195  4
; 1134          1196  4             LOCAL
; 1135          1197  4                 RSZ     : WORD;
; 1136          1198  4
; 1137          1199  4             RM$KEY_DESC(0);
; 1138          1200  4
; 1139          1201  4             ! Add record overhead, and calculate record's size.
; 1140          1202  4             !
; 1141          1203  5             BEGIN
; 1142          1204  5
; 1143          1205  5             LOCAL
; 1144          1206  5                 REC_SIZE,
; 1145          1207  5                 RECORD_OVHD;
; 1146          1208  5
; 1147          1209  5             RECORD_OVHD = RM$REC_OVHD(0; REC_SIZE);
; 1148          1210  5
; 1149          1211  5             ! If this primary data record had been updated within a Recovery
; 1150          1212  5             ! Unit then retrieve its true size from the last two bytes of the
; 1151          1213  5             ! reserved space.
; 1152          1214  5             !
; 1153          1215  5             IF .REC_ADDR[IRC$V_RU_UPDATE]
; 1154          1216  5             THEN
```

```
; 1155   1217  6            RSZ = .(.REC_ADDR + .RECORD_OVHD
; 1156   1218  6                                + .REC_SIZE
; 1157   1219  5                                - IRC$C_DATSZFLD)<0,16>
; 1158   1220  5        ELSE
; 1159   1221  5            RSZ = .REC_SIZE;
; 1160   1222  5
; 1161   1223  5        REC_ADDR = .REC_ADDR + .RECORD_OVHD;
; 1162   1224  4        END;
; 1163   1225  4
; 1164   1226  4        ! It will only be necessary to unpack the primary data record
; 1165   1227  4        ! if its a prologue 3 file.
; 1166   1228  4        !
; 1167   1229  5        IF  (.IFAB[IFB$B_PLG_VER] EQLU PLG$C_VER_3)
; 1168   1230  4        THEN
; 1169   1231  5            BEGIN
; 1170   1232  5
; 1171   1233  5            GLOBAL REGISTER
; 1172   1234  5                R_BKT_ADDR;
; 1173   1235  5
; 1174   1236  5            ! If the key of reference is the primary key then signal
; 1175   1237  5            ! RM$UNPACK_REC that the primary key for this data record maybe
; 1176   1238  5            ! found in expanded form within keybuffer 2.
; 1177   1239  5            !
; 1178   1240  5            IF .IRAB[IRB$B_RP_KREF] EQLU 0
; 1179   1241  5            THEN
; 1180   1242  5                AP = 2
; 1181   1243  5            ELSE
; 1182   1244  5                AP = 0;
; 1183   1245  5
; 1184   1246  5            RSZ = RM$UNPACK_REC(.IRAB[IRB$L_RECBUF],.RSZ);
; 1185   1247  4            END;
; 1186   1248  4
; 1187   1249  6        IF (.RAB[RAB$V_LOC] AND NOT (.IFAB[IFB$V_UPD])
; 1188   1250  5            AND
; 1189   1251  5            NOT (.BBLOCK[.IRAB[IRB$L_CURBDB], BDB$V_NOLOCATE]))
; 1190   1252  4        THEN
; 1191   1253  5            BEGIN                           ! We can do locate mode get
; 1192   1254  5            IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
; 1193   1255  5            THEN
; 1194   1256  5                RAB[RAB$L_RBF] = .REC_ADDR
; 1195   1257  5            ELSE
; 1196   1258  5                RAB[RAB$L_RBF] = .IRAB[IRB$L_RECBUF];
; 1197   1259  5            RAB[RAB$W_RSZ] = .RSZ;
; 1198   1260  5            END
; 1199   1261  4        ELSE
; 1200   1262  5            BEGIN                           ! We must do move mode get
; 1201   1263  5
; 1202   1264  5            LOCAL
; 1203   1265  5                USZ : WORD;
; 1204   1266  5
; 1205   1267  5            USZ = .RAB[RAB$W_USZ];
; 1206   1268  5
; 1207   1269  5            IF .USZ EQL 0
; 1208   1270  5            THEN
; 1209   1271  6                BEGIN
; 1210   1272  6
; 1211   1273  6                IF .FLAGS[AP_STATUS]
```

```
; 1212   1274  6                THEN
; 1213   1275  6                    IRAB[IRB$V_UNLOCK_RP] = 1;       ! flag unlock RP
; 1214   1276  6
; 1215   1277  6                STATUS = RMSERR(USZ);
; 1216   1278  6                END
; 1217   1279  5            ELSE
; 1218   1280  6                BEGIN
; 1219   1281  6
; 1220   1282  6                LOCAL
; 1221   1283  6                    UBF_ADDR;
; 1222   1284  6                UBF_ADDR = .RAB [RAB$L_UBF];
; 1223   1285  6
; 1224   1286  6                IF .RSZ GTRU .USZ
; 1225   1287  6                THEN
; 1226   1288  7                    BEGIN
; 1227   1289  7                    RAB[RAB$L_STV] = .RSZ;
; 1228   1290  7                    RSZ = .USZ;
; 1229   1291  7                    STATUS = RMSERR(RTB);
; 1230   1292  6                    END;
; 1231   1293  6
; 1232   1294  6                IF RM$NOWRT_LONG(.RSZ, .UBF_ADDR, .IRAB[IRB$B_MODE])
; 1233   1295  6                THEN
; 1234   1296  7                    BEGIN
; 1235   1297  7
; 1236   1298  7                    IF .FLAGS[AP_STATUS]
; 1237   1299  7                    THEN
; 1238   1300  7                        IRAB[IRB$V_UNLOCK_RP] = 1;  ! flag unlock RP
; 1239   1301  7
; 1240   1302  7                    STATUS = RMSERR(UBF);
; 1241   1303  7                    END
; 1242   1304  6                ELSE
; 1243   1305  7                    BEGIN
; 1244   1306  7                    RAB[RAB$W_RSZ] = .RSZ;
; 1245   1307  7                    RAB[RAB$L_RBF] = .UBF_ADDR;
; 1246   1308  7                    IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
; 1247   1309  7                    THEN
; 1248   1310  7                        CH$MOVE(.RSZ, .REC_ADDR, .UBF_ADDR)
; 1249   1311  7                    ELSE
; 1250   1312  7                        CH$MOVE(.RSZ, .IRAB[IRB$L_RECBUF], .UBF_ADDR );
; 1251   1313  7
; 1252   1314  6                    END;
; 1253   1315  6
; 1254   1316  5                END;
; 1255   1317  5
; 1256   1318  4            END;
; 1257   1319  4
; 1258   1320  3        END;
; 1259   1321  3
; 1260   1322  2    END;
; 1261   1323  2
; 1262   1324  3  IF (.STATUS
; 1263   1325  3      OR
; 1264   1326  3      (.STATUS<0, 16> EQL RMSERR(RTB)))
; 1265   1327  2  THEN
; 1266   1328  3      BEGIN
; 1267   1329  3
; 1268   1330  3      IF NOT .IRAB[IRB$V_FIND]
```

```
; 1269    1331  3        THEN
; 1270    1332  4            BEGIN
; 1271    1333  4
; 1272    1334  4            ! Set up the next record context for all $GET operations.
; 1273    1335  4            !
; 1274    1336  4            SETUP_NRP_DATA();
; 1275    1337  4
; 1276    1338  4            ! If dupes aren't allowed on primary key, save primary key value of
; 1277    1339  4            ! this record in keybuffer 3 so reaccess and record lock window can
; 1278    1340  4            ! be avoided on subsequent random FIND on primary key (cobol does
; 1279    1341  4            ! this a lot).
; 1280    1342  4            !
; 1281    1343  4            IF NOT .IDX_DFN[IDX$V_DUPKEYS]
; 1282    1344  4            THEN
; 1283    1345  5                BEGIN
; 1284    1346  5
; 1285    1347  5                LOCAL
; 1286    1348  5                    TMP_REC_ADDR;
; 1287    1349  5
; 1288    1350  5                TMP_REC_ADDR = .REC_ADDR;
; 1289    1351  5                REC_ADDR = .RAB [RAB$L_RBF];
; 1290    1352  5
; 1291    1353  5                IF RM$NOREAD_LONG ( .RAB [RAB$W_RSZ], .REC_ADDR,
; 1292    1354  5                        .IRAB [IRB$B_MODE] )
; 1293    1355  5                THEN
; 1294    1356  5                    STATUS = RMSERR(RBF);
; 1295    1357  5                AP = 3;                 ! no overhead / expanded
; 1296    1358  6                IF (.STATUS
; 1297    1359  6                   OR
; 1298    1360  6                   (.STATUS<0, 16> EQL RMSERR(RTB)))
; 1299    1361  5                THEN
; 1300    1362  6                    BEGIN
; 1301    1363  6
; 1302    1364  6                    GLOBAL REGISTER
; 1303    1365  6                        R_BDB;
; 1304    1366  6
; 1305    1367  6                    RM$RECORD_KEY (KEYBUF_ADDR(3));
; 1306    1368  5                    END;
; 1307    1369  5                REC_ADDR = .TMP_REC_ADDR
; 1308    1370  4                END;
; 1309    1371  3            END;
; 1310    1372  3
; 1311    1373  3        END
; 1312    1374  2    ELSE
; 1313    1375  3        BEGIN
; 1314    1376  3
; 1315    1377  3        ! UNLOCK_RP is used as a flag on error conditions to indicate whether
; 1316    1378  3        ! the record described by the current record (rp) information is to be
; 1317    1379  3        ! unlocked or not.  This will be the case when buffer errors are
; 1318    1380  3        ! discovered after the new record has been locked, or if the current
; 1319    1381  3        ! record before this operation was not unlocked at the beginning of
; 1320    1382  3        ! this operation. In both cases the RFA of the record to unlock will be
; 1321    1383  3        ! found in IRB$L_NEXT_VBN and IRB$W_NEXT_ID. In the former case,
; 1322    1384  3        ! because these fields contain the RFA of the "next" record, and in the
; 1323    1385  3        ! latter, because the only reason why the current record was not
; 1324    1386  3        ! unlocked at the beginning of the operation because it itself was
; 1325    1387  3        ! being retrieved by the operation, and if this was the case then
```

```
; 1326          1388  3          ! these same fields would contain the RFA of the current record from
; 1327          1389  3          ! when it had been originally locked.
; 1328          1390  3          !
; 1329          1391  3          IF TESTBITSC(IRAB[IRB$V_UNLOCK_RP])
; 1330          1392  3          THEN
; 1331          1393  3              RM$UNLOCK (.IRAB[IRB$L_NEXT_VBN], .IRAB[IRB$W_NEXT_ID]);
; 1332          1394  3
; 1333          1395  3          ! If end-of-file has been reached, set the corresponding IRAB bit.
; 1334          1396  3          !
; 1335          1397  4          IF .STATUS EQL RMSERR(EOF)
; 1336          1398  3          THEN
; 1337          1399  3              IRAB[IRB$V_EOF] = 1;
; 1338          1400  3
; 1339          1401  3          ! There is no longer a current primary data record.
; 1340          1402  3          !
; 1341          1403  3          IRAB[IRB$L_UDR_VBN] = 0;
; 1342          1404  3          IRAB[IRB$W_UDR_ID]  = 0;
; 1343          1405  2          END;
; 1344          1406  2
; 1345          1407  2      RAB[RAB$L_RFA0] = .IRAB[IRB$L_UDR_VBN];
; 1346          1408  2      RAB[RAB$W_RFA4] = .IRAB[IRB$W_UDR_ID];
; 1347          1409  2
; 1348          1410  2      IRAB[IRB$V_FIND] = 0;
; 1349          1411  3      BEGIN
; 1350          1412  3
; 1351          1413  3      GLOBAL REGISTER
; 1352          1414  3          R_BDB;
; 1353          1415  3
; 1354          1416  3      IF (BDB = .IRAB[IRB$L_CURBDB]) NEQ 0
; 1355          1417  3      THEN
; 1356          1418  3          RM$RLSBKT(0);
; 1357          1419  3
; 1358          1420  3      IRAB[IRB$L_CURBDB] = 0;
; 1359          1421  2      END;
; 1360          1422  2      RETURN .STATUS
; 1361          1423  2
; 1362          1424  1      END;


                                    00FC    8F  BB 00000 RM$GET3B::
                                                                 PUSHR   #^M<R2,R3,R4,R5,R6,R7>          ; 0876
                                5E              10  C2 00004         SUBL2   #16, SP                         ;
                                                57  D4 00007         CLRL    IDX_DFN                         ; 0996
                                          04    AE  D4 00009         CLRL    FLAGS                           ; 0997
                                                50  D4 0000C 1$:     CLRL    R0                              ; 1005
                    02          04  AE          02  E5 0000E         BBCC    #2, FLAGS, 2$                   ;
                                                50  D6 00013         INCL    R0                              ;
51          04      AE              01          01  EF 00015 2$:     EXTZV   #1, #1, FLAGS, R1               ;
                    7E              50          51  C9 0001B         BISL3   R1, R0, -(SP)                   ;
                                                  0000V 30 0001F     BSBW    GET_RECORD                      ; 1004
                                5E              04  C0 00022         ADDL2   #4, SP                          ;
                                6E              50  D0 00025         MOVL    R0, STATUS                      ;
                                01              6E  B1 00028         CMPW    STATUS, #1                      ; 1010
                                                6C  13 0002B         BEQL    10$                             ;
```

```
                                    57  D5 0002D          TSTL    IDX_DFN                          ; 1018
                                    68  13 0002F          BEQL    10$
                                    50  D4 00031          CLRL    R0                               ; 1022
                        8262   8F   6E  B1 00033          CMPW    STATUS, #33378
                                    04  12 00033          BNEQ    3$
                                    50  D6 0003A          INCL    R0
                                    04  11 0003C          BRB     4$
                                    6E  D5 0003E 3$:      TSTL    STATUS                           ; 1024
                                    06  12 00040          BNEQ    5$
                           02    1E   A8  91 00042 4$:    CMPB    30(RAB), #2                      ; 1026
                                    0C  12 00046          BNEQ    6$
                                 21   A7  95 00048 5$:    TSTB    33(IDX_DFN)                      ; 1028
                                    4C  13 0004B          BEQL    10$
                        8061   8F   6E  B1 0004D          CMPW    STATUS, #32865                   ; 1030
                                    45  12 00052          BNEQ    10$
                           1F       50  E9 00054 6$:      BLBC    R0, 7$                           ; 1048
                                 21   A7  95 00057          TSTB    33(IDX_DFN)                      ; 1050
                                    1A  12 0005A          BNEQ    7$
                                 1E   A8  95 0005C          TSTB    30(RAB)                          ; 1052
                                    15  13 0005F          BEQL    7$
                           11    1C   A7  E8 00061          BLBS    28(IDX_DFN), 7$                  ; 1054
                     0C    06   A8      05  E0 00065          BBS     #5, 6(RAB), 7$                   ; 1056
                     07    06   A8      06  E0 0006A          BBS     #6, 6(RAB), 7$
                           6E  82B2   8F  3C 0006F          MOVZWL  #33458, STATUS                   ; 1059
                                    23  11 00074          BRB     10$                              ; 1058
                                    6E  D5 00076 7$:      TSTL    STATUS                           ; 1070
                                    06  13 00078          BEQL    8$
                        04   AE     02  88 0007A          BISB2   #2, FLAGS                        ; 1072
                                    04  11 0007E          BRB     9$
                        04   AE     04  88 00080 8$:      BISB2   #4, FLAGS                        ; 1074
                           51    20   A7  9A 00084 9$:    MOVZBL  32(IDX_DFN), R1                  ; 1076
                           50  00B4   CA  3C 00088          MOVZWL  180(IFAB), R0
                           50    60   A9  C0 0008D          ADDL2   96(IRAB), R0
                     60    60   B9      51  28 00091          MOVC3   R1, @96(IRAB), (R0)
                                  FF73  31 00096          BRW     1$                               ; 0999
04   AE               01           00     5C  F0 00099 10$:    INSV    AP, #0, #1, FLAGS                ; 1083
                     0A    04   AE      01  E1 0009F          BBC     #1, FLAGS, 11$                   ; 1090
                           01          6E  B1 000A4          CMPW    STATUS, #1                       ; 1092
                                    05  12 000A7          BNEQ    11$
                           6E  8061   8F  3C 000A9          MOVZWL  #32865, STATUS                   ; 1094
                        04   A9     20  8A 000AE 11$:    BICB2   #32, 4(IRAB)                     ; 1096
                        58   A9    28   A8  D0 000B2          MOVL    40(RAB), 88(IRAB)                ; 1100
                        56   A9    22   A8  B0 000B7          MOVW    34(RAB), 86(IRAB)                ; 1101
                           5C          6E  E9 000BC          BLBC    STATUS, 17$                      ; 1107
                                    1E  A8  95 000BF          TSTB    30(RAB)                          ; 1109
                                    57  12 000C2          BNEQ    17$
                     52    05   A8      06  E1 000C4          BBC     #6, 5(RAB), 17$                  ; 1111
                           50    34   A8  9A 000C9          MOVZBL  52(RAB), KEYSIZE                 ; 1119
                                    16  12 000CD          BNEQ    14$                              ; 1121
                                    1D  A7  95 000CF          TSTB    29(IDX_DFN)                      ; 1124
                                    06  13 000D2          BEQL    12$
                           01    1E   A7  91 000D4          CMPB    30(IDX_DFN), #1                  ; 1125
                                    07  1B 000D8          BLEQU   13$
                           6E  85A4   8F  3C 000DA 12$:    MOVZWL  #34212, STATUS                   ; 1127
                                    04  11 000DF          BRB     14$
                           50    20   A7  9A 000E1 13$:    MOVZBL  32(IDX_DFN), KEYSIZE             ; 1129
                        20   A7     50  91 000E5 14$:    CMPB    KEYSIZE, 32(IDX_DFN)             ; 1136
```

```
                                05  1B 000E9          BLEQU   15$                                          ;
                    6E    85A4   8F  3C 000EB          MOVZWL  #34212, STATUS                               ; 1138
                    53      30   A8  D0 000F0 15$:     MOVL    48(RAB), KBF_ADDR                            ; 1141
            63      50      0A   A9  0C 000F4          PROBER  10(IRAB), KEYSIZE, (KBF_ADDR)                ; 1143
                                05  12 000F9          BNEQ    16$                                          ;
                    6E    858C   8F  3C 000FB          MOVZWL  #34188, STATUS                               ;
                    18           6E  E9 00100 16$:     BLBC    STATUS, 17$                                  ; 1145
                    5C           03  D0 00103          MOVL    #3, AP                                       ; 1148
                    51    00B4   CA  3C 00106          MOVZWL  180(IFAB), R1                                ; 1154
                    51      60   A9  C0 0010B          ADDL2   96(IRAB), R1                                 ;
                          0000G  30  0010F          BSBW    RM$COMPARE_KEY                               ;
                               50  D5 00112          TSTL    R0                                           ;
                                05  18 00114          BGEQ    17$                                          ;
                    6E    8051   8F  3C 00116          MOVZWL  #32849, STATUS                               ; 1156
                    03           6E  E8 0011B 17$:     BLBS    STATUS, 18$                                  ; 1161
                          010A  31  0011E          BRW     38$                                          ;
            1D      05    A9     01  E1 00121 18$:     BBC     #1, 5(IRAB), 21$                             ; 1165
                    04    A9     20  88 00126          BISB2   #32, 4(IRAB)                                 ; 1172
                            1E   A8  95 0012A          TSTB    30(RAB)                                      ; 1176
                                05  12 0012D          BNEQ    19$                                          ;
                          FE71  30  0012F          BSBW    SETUP_NRP_DATA                               ; 1178
                                0D  11 00132          BRB     20$                                          ;
                00B0    C9      78   A9  D0 00134 19$:     MOVL    120(IRAB), 176(IRAB)                         ; 1185
                00BC    C9    0080   C9  B0 0013A          MOVW    128(IRAB), 188(IRAB)                         ; 1186
                                72  11 00141 20$:     BRB     29$                                          ; 1165
                                7E  D4 00143 21$:     CLRL    -(SP)                                        ; 1199
                          0000G  30  00145          BSBW    RM$KEY_DESC                                  ;
                    5E           04  C0 00148          ADDL2   #4, SP                                       ;
                                51  D4 0014B          CLRL    R1                                           ; 1209
                          0000G  30  0014D          BSBW    RM$REC_OVHD                                  ;
            0E      66           06  E1 00150          BBC     #6, (REC_ADDR), 22$                          ; 1215
            52      56           50  C1 00154          ADDL3   RECORD_OVHD, REC_ADDR, R2                    ; 1217
                            FE A142  9F 00158          PUSHAB  -2(REC_SIZE)[R2]                             ;
                08  AE           9E  B0 0015C          MOVW    @(SP)+, RSZ                                  ;
                                04  11 00160          BRB     23$                                          ;
                08  AE           51  B0 00162 22$:     MOVW    REC_SIZE, RSZ                                ; 1221
                    56           50  C0 00166 23$:     ADDL2   RECORD_OVHD, REC_ADDR                        ; 1223
                    03    00B7   CA  91 00169          CMPB    183(IFAB), #3                                ; 1229
                                1C  12 0016E          BNEQ    26$                                          ;
                          00C2  C9  95 00170          TSTB    194(IRAB)                                    ; 1240
                                05  12 00174          BNEQ    24$                                          ;
                    5C           02  D0 00176          MOVL    #2, AP                                       ; 1242
                                02  11 00179          BRB     25$                                          ;
                                5C  D4 0017B 24$:     CLRL    AP                                           ; 1244
                    51      08   AE  3C 0017D 25$:     MOVZWL  RSZ, R1                                      ; 1246
                    50      68   A9  D0 00181          MOVL    104(IRAB), R0                                ;
                          0000G  30  00185          BSBW    RM$UNPACK_REC                                ;
                08  AE           50  B0 00188          MOVW    R0, RSZ                                      ;
                    27      06   A8  E9 0018C 26$:     BLBC    6(RAB), 30$                                  ; 1249
            22  22  AA           03  E0 00190          BBS     #3, 34(IFAB), 30$                            ;
                    50      20   A9  D0 00195          MOVL    32(IRAB), R0                                 ; 1251
            19  0A  A0           04  E0 00199          BBS     #4, 10(R0), 30$                              ;
                    03    00B7   CA  91 0019E          CMPB    183(IFAB), #3                                ; 1254
                                06  1E 001A3          BGEQU   27$                                          ;
                28  A8           56  D0 001A5          MOVL    REC_ADDR, 40(RAB)                            ; 1256
                                05  11 001A9          BRB     28$                                          ;
                28  A8      68   A9  D0 001AB 27$:     MOVL    104(IRAB), 40(RAB)                           ; 1258
```

```
                22  A8      08  AE  B0 001B0 28$:   MOVW    RSZ, 34(RAB)                        ; 1259
                                71  11 001B5 29$:   BRB     37$                                 ; 1249
                    50      20  A8  B0 001B7 30$:   MOVW    32(RAB), USZ                        ; 1267
                                0F  12 001BB        BNEQ    32$                                 ; 1269
                    04      04  AE  E9 001BD        BLBC    FLAGS, 31$                          ; 1273
                05  A9          20  88 001C1        BISB2   #32, 5(IRAB)                        ; 1275
                    6E    86F4  8F  3C 001C5 31$:   MOVZWL  #34548, STATUS                      ; 1277
                                5C  11 001CA        BRB     37$                                 ; 1269
                0C  AE      24  A8  D0 001CC 32$:   MOVL    36(RAB), UBF_ADDR                   ; 1284
                    50      08  AE  B1 001D1        CMPW    RSZ, USZ                            ; 1286
                                0E  1B 001D5        BLEQU   33$                                 ;
                0C  A8      08  AE  3C 001D7        MOVZWL  RSZ, 12(RAB)                        ; 1289
                08  AE          50  B0 001DC        MOVW    USZ, RSZ                            ; 1290
                    6E    81A8  8F  3C 001E0        MOVZWL  #33192, STATUS                      ; 1291
                    7E      0A  A9  9A 001E5 33$:   MOVZBL  10(IRAB), -(SP)                     ; 1294
                            10  AE  DD 001E9        PUSHL   UBF_ADDR                            ;
                    7E      10  AE  3C 001EC        MOVZWL  RSZ, -(SP)                          ;
                            0000G 30 001F0          BSBW    RM$NOWRT_LONG                       ;
                    5E          0C  C0 001F3        ADDL2   #12, SP                             ;
                    0F          50  E9 001F6        BLBC    R0, 35$                             ;
                    04      04  AE  E9 001F9        BLBC    FLAGS, 34$                          ; 1298
                05  A9          20  88 001FD        BISB2   #32, 5(IRAB)                        ; 1300
                    6E    86EC  8F  3C 00201 34$:   MOVZWL  #34540, STATUS                      ; 1302
                                20  11 00206        BRB     37$                                 ; 1294
                22  A8      08  AE  B0 00208 35$:   MOVW    RSZ, 34(RAB)                        ; 1306
                28  A8      0C  AE  D0 0020D        MOVL    UBF_ADDR, 40(RAB)                   ; 1307
                    03    00B7  CA  91 00212        CMPB    183(IFAB), #3                       ; 1308
                                08  1E 00217        BGEQU   36$                                 ;
            0C  BE      66      08  AE  28 00219    MOVC3   RSZ, (REC_ADDR), @UBF_ADDR          ; 1310
                                07  11 0021F        BRB     37$                                 ;
            0C  BE  68  B9      08  AE  28 00221 36$:   MOVC3   RSZ, @104(IRAB), @UBF_ADDR      ; 1312
                    07          6E  E8 00228 37$:   BLBS    STATUS, 39$                         ; 1324
                81A8  8F        6E  B1 0022B 38$:   CMPW    STATUS, #33192                      ; 1326
                                4C  12 00230        BNEQ    43$                                 ;
            6D  05  A9          01  E0 00232 39$:   BBS     #1, 5(IRAB), 46$                    ; 1330
                            FD69  30 00237          BSBW    SETUP_NRP_DATA                      ; 1336
                    66      1C  A7  E8 0023A        BLBS    28(IDX_DFN), 46$                    ; 1343
                    52          56  D0 0023E        MOVL    REC_ADDR, TMP_REC_ADDR              ; 1350
                    56      28  A8  D0 00241        MOVL    40(RAB), REC_ADDR                   ; 1351
                    7E      0A  A9  9A 00245        MOVZBL  10(IRAB), -(SP)                     ; 1354
                                56  DD 00249        PUSHL   REC_ADDR                            ; 1353
                    7E      22  A8  3C 0024B        MOVZWL  34(RAB), -(SP)                      ;
                            0000G 30 0024F          BSBW    RM$NOREAD_LONG                      ;
                    5E          0C  C0 00252        ADDL2   #12, SP                             ;
                    05          50  E9 00255        BLBC    R0, 40$                             ;
                    6E    8654  8F  3C 00258        MOVZWL  #34388, STATUS                      ; 1356
                    5C          03  D0 0025D 40$:   MOVL    #3, AP                              ; 1357
                    07          6E  E8 00260        BLBS    STATUS, 41$                         ; 1358
                81A8  8F        6E  B1 00263        CMPW    STATUS, #33192                      ; 1360
                                0F  12 00268        BNEQ    42$                                 ;
                    50    00B4  CA  3C 0026A 41$:   MOVZWL  180(IFAB), R0                       ; 1367
                          60 B940 3F 0026F          PUSHAW  @96(IRAB)[R0]                       ;
                            0000G 30 00273          BSBW    RM$RECORD_KEY                       ;
                    5E          04  C0 00276        ADDL2   #4, SP                              ;
                    56          52  D0 00279 42$:   MOVL    TMP_REC_ADDR, REC_ADDR              ; 1369
                                26  11 0027C        BRB     46$                                 ; 1324
            0C      04  A9      0D  E5 0027E 43$:   BBCC    #13, 4(IRAB), 44$                   ; 1391
```

```
                        52    0080   C9  3C 00283          MOVZWL  128(IRAB), R2                          ; 1393
                        51      78   A9  D0 00288          MOVL    120(IRAB), R1                          ;
                                   0000G  30 0028C          BSBW    RM$UNLOCK                              ;
                0000827A   8F         6E  D1 0028F 44$:     CMPL    STATUS, #33402                         ; 1397
                                      04  12 00296          BNEQ    45$                                    ;
                        04   A9       02  88 00298          BISB2   #2, 4(IRAB)                            ; 1399
                             00B0     C9  D4 0029C 45$:     CLRL    176(IRAB)                              ; 1403
                             00BC     C9  B4 002A0          CLRW    188(IRAB)                              ; 1404
                        10   A8  00B0 C9  D0 002A4 46$:     MOVL    176(IRAB), 16(RAB)                     ; 1407
                        14   A8  00BC C9  B0 002AA          MOVW    188(IRAB), 20(RAB)                     ; 1408
                        05   A9       02  8A 002B0          BICB2   #2, 5(IRAB)                            ; 1410
                        54      20    A9  D0 002B4          MOVL    32(IRAB), BDB                          ; 1416
                                      08  13 002B8          BEQL    47$                                    ;
                                      7E  D4 002BA          CLRL    -(SP)                                  ; 1418
                                   0000G  30 002BC          BSBW    RM$RLSBKT                              ;
                        5E            04  C0 002BF          ADDL2   #4, SP                                 ;
                                      20  A9 D4 002C2 47$:  CLRL    32(IRAB)                               ; 1420
                        50            8E  D0 002C5          MOVL    STATUS, R0                             ; 1422
                        5E            0C  C0 002C8          ADDL2   #12, SP                                ; 1424
                             00FC     8F  BA 002CB          POPR    #^M<R2,R3,R4,R5,R6,R7>                 ;
                                          05 002CF          RSB                                            ;

; Routine Size:  720 bytes,    Routine Base:  RM$RMS3 + 005D

; 1363          1425  1
```

```
; 1365          1426  1 %SBTTL 'GET_RECORD'
; 1366          1427  1 ROUTINE GET_RECORD (REPOS_STATUS) : L_GET_RECORD =
; 1367          1428  1
; 1368          1429  1 !++
; 1369          1430  1 !
; 1370          1431  1 ! FUNCTIONAL DESCRIPTION:
; 1371          1432  1 !
; 1372          1433  1 !       This routine implements the actual retrieval of the
; 1373          1434  1 !       data record for internal RMS usage. The use request
; 1374          1435  1 !       is checked for valid input parameters and all internal state
; 1375          1436  1 !       information is setup to retrieve the record.
; 1376          1437  1 !       Then current record is unlocked if required and the
; 1377          1438  1 !       requested record retrieved and locked if required. All NRP
; 1378          1439  1 !       update data is saved in the IRAB but is not placed in the
; 1379          1440  1 !       NRP fields of the IRAB.
; 1380          1441  1 !
; 1381          1442  1 ! CALLING SEQUENCE:
; 1382          1443  1 !
; 1383          1444  1 !       GET_RECORD()
; 1384          1445  1 !
; 1385          1446  1 ! INPUT PARAMETERS:
; 1386          1447  1 !
; 1387          1448  1 !       REPOS_STATUS    - if 1, then RMS is performing a re-positioning.
; 1388          1449  1 !
; 1389          1450  1 ! IMPLICIT INPUTS:
; 1390          1451  1 !
; 1391          1452  1 !       Same as for RM$GET3 or RM$FIND3
; 1392          1453  1 !
; 1393          1454  1 ! OUTPUT PARAMETERS:
; 1394          1455  1 !
; 1395          1456  1 !       IRAB context setup for retrieved record:
; 1396          1457  1 !
; 1397          1458  1 !               CURBDB,RFA_VBN,RFA_ID,SAVE_POS,FIRST_VBN
; 1398          1459  1 !               FIRST_ID,REC_ADDR,NEXT_VBN,NEXT_ID
; 1399          1460  1 !
; 1400          1461  1 !       RAB The DCT field is cleared in all cases.
; 1401          1462  1 !
; 1402          1463  1 !       If the value of the routine is a success status then
; 1403          1464  1 !       the AP = 0 if no special action is needed to unlock the RP
; 1404          1465  1 !       and is 1 if special action is needed, on errors detected after
; 1405          1466  1 !       this routine.
; 1406          1467  1 !
; 1407          1468  1 ! IMPLICIT OUTPUTS:
; 1408          1469  1 !
; 1409          1470  1 !       IRB$V_UNLOCK_RP
; 1410          1471  1 !       IRB$V_FIND_LAST = 0
; 1411          1472  1 !       IRB$B_RP_KREF
; 1412          1473  1 !
; 1413          1474  1 ! ROUTINE VALUE:
; 1414          1475  1 !
; 1415          1476  1 !       Internal RMS status code
; 1416          1477  1 !
; 1417          1478  1 ! SIDE EFFECTS:
; 1418          1479  1 !
; 1419          1480  1 !       Retrieved record maybe locked.
; 1420          1481  1 !       Old current record may have been unlocked.
; 1421          1482  1 !       The data bucket for the retrieved record is accessed.
```

```
; 1422          1483  1 !       NLK is ignored if the process is in a Recovery Unit.
; 1423          1484  1 !       If a primary data record is found to have been deleted within a Recovery
; 1424          1485  1 !           Unit it might be deleted for good.
; 1425          1486  1 !       If a primary data record is found to have been updated within a Recovery
; 1426          1487  1 !           Unit it might be re-formatted.
; 1427          1488  1 !
; 1428          1489  1 !--
; 1429          1490  1
; 1430          1491  2     BEGIN
; 1431          1492  2
; 1432          1493  2     BUILTIN
; 1433          1494  2         TESTBITCC,
; 1434          1495  2         TESTBITSC,
; 1435          1496  2         AP;
; 1436          1497  2
; 1437          1498  2     EXTERNAL REGISTER
; 1438          1499  2         R_REC_ADDR_STR,
; 1439          1500  2         R_IDX_DFN_STR,
; 1440          1501  2         COMMON_RAB_STR;
; 1441          1502  2
; 1442          1503  2     LABEL
; 1443          1504  2         UNLOCK,
; 1444          1505  2         KEY;
; 1445          1506  2
; 1446          1507  2     IRAB[IRB$B_CACHEFLGS] = 0;
; 1447          1508  2     IRAB[IRB$W_SRCHFLAGS] = 0;
; 1448          1509  2     IRAB[IRB$V_DUP] = 0;
; 1449          1510  2
; 1450          1511  2     ! Based on the record access mode (RAC) of this operation (GET/FIND)
; 1451          1512  2     ! set up the IRAB RP fields, key buffer 2 etc to retrieve the record.
; 1452          1513  2     !
; 1453          1514  2     !
; 1454          1515  2     RAB[RAB$L_DCT] = 0;
; 1455          1516  2
; 1456          1517  2     ! Get record block 1 --- set up the IRAB search context data to get the
; 1457          1518  2     ! record the user is requesting, and unlock the current record if this is
; 1458          1519  2     ! required.
; 1459          1520  2     !
; 1460          1521  2 UNLOCK :
; 1461          1522  3     BEGIN
; 1462          1523  3
; 1463          1524  3     CASE .RAB[RAB$B_RAC] FROM RAB$C_SEQ TO RAB$C_RFA OF
; 1464          1525  3         SET
; 1465          1526  3
; 1466          1527  3         [RAB$C_SEQ] :
; 1467          1528  3
; 1468          1529  3             !+
; 1469          1530  3             ! Sequential Access:
; 1470          1531  3             !
; 1471          1532  3             !   Setup to retrieve the record associated with the NRP if this
; 1472          1533  3             !   is a GET and the last operation was not a FIND or if this is
; 1473          1534  3             !   a FIND.
; 1474          1535  3             !
; 1475          1536  3             !   If last operation was a FIND and this operation is a GET then
; 1476          1537  3             !   retrieve record which is described by the NRP if that FIND was
; 1477          1538  3             !   sequential.
; 1478          1539  3             !
```

```
; 1479          1540  3          !   If the FIND was random then change the NRP data for the record
; 1480          1541  3          !   which was found and retrieve it.
; 1481          1542  3          !
; 1482          1543  3          !   Note that a sequential FIND following a random FIND returns to
; 1483          1544  3          !   the sequential next record (NRP).  That is to say that the
; 1484          1545  3          !   random operation will not change the NRP VBN and ID fields.
; 1485          1546  3          !-
; 1486          1547  4          BEGIN
; 1487          1548  4
; 1488          1549  4          IF  .IRAB[ IRB$V_CON_EOF ]
; 1489          1550  4          THEN
; 1490          1551  4              RETURN RMSERR(EOF);
; 1491          1552  4
; 1492          1553  5          IF (.IRAB[IRB$V_FIND_LAST]
; 1493          1554  5              AND
; 1494          1555  5              NOT (.IRAB[IRB$V_FIND]))
; 1495          1556  4          THEN
; 1496          1557  5              BEGIN
; 1497          1558  5
; 1498          1559  5              ! NOTE: keybuffer 2 contains key value, RP_KREF has key of
; 1499          1560  5              ! reference, RP_VBN and RP_ID contains record's RFA/RRV, and
; 1500          1561  5              ! SAVE_DUP contains the duplicate position count.
; 1501          1562  5              !
; 1502          1563  5              IF TESTBITSC(IRAB[IRB$V_SKIP_NEXT])
; 1503          1564  5              THEN
; 1504          1565  5
; 1505          1566  5                  ! Last find was sequential so retrieve the record described
; 1506          1567  5                  ! by the NRP unless RMS is already positioned at the end
; 1507          1568  5                  ! of the file.
; 1508          1569  5                  !
; 1509          1570  5                  IF .IRAB[IRB$V_EOF]
; 1510          1571  5                  THEN
; 1511          1572  6                      RETURN RMSERR (EOF)
; 1512          1573  5                  ELSE
; 1513          1574  5                      RM$KEY_DESC(.IRAB[IRB$B_CUR_KREF])
; 1514          1575  5              ELSE
; 1515          1576  5
; 1516          1577  5                  ! Last operation was a find random and this operation is a
; 1517          1578  5                  ! get sequential. Setup the local NRP context so that this
; 1518          1579  5                  ! same record will be retrieved. Since there has been no
; 1519          1580  5                  ! intervening operation, calling SETUP_NRP_DATA will
; 1520          1581  5                  ! accomplish this.
; 1521          1582  5                  !
; 1522          1583  6                  BEGIN
; 1523          1584  6                  SETUP_NRP_DATA();
; 1524          1585  6                  RM$KEY_DESC(.IRAB[IRB$B_RP_KREF]);
; 1525          1586  5                  END;
; 1526          1587  5
; 1527          1588  5              IRAB[IRB$B_KEYSZ] = .IDX_DFN[IDX$B_KEYSZ];
; 1528          1589  5
; 1529          1590  5              ! Unless no lock is desired on this record (and the process
; 1530          1591  5              ! is not within a Recovery Unit), leave this block to avoid
; 1531          1592  5              ! unlocking the current record.  This avoids a potential
; 1532          1593  5              ! window where the record is unlocked as it is reaccessed on
; 1533          1594  5              ! this get operation.
; 1534          1595  5              !
; 1535          1596  5              IF NOT .RAB[RAB$V_NLK]
```

```
; 1536          1597  5                 OR
; 1537          1598  5                 .IFAB[IFB$V_RUP]
; 1538          1599  5             THEN
; 1539          1600  5                 LEAVE UNLOCK;
; 1540          1601  5
; 1541          1602  5             END
; 1542          1603  4         ELSE
; 1543          1604  5             BEGIN
; 1544          1605  5
; 1545          1606  5             ! Return immediately if already at end-of-file.
; 1546          1607  5             !
; 1547          1608  5             IF .IRAB[IRB$V_EOF]
; 1548          1609  5             THEN
; 1549          1610  5                 RETURN RMSERR(EOF);
; 1550          1611  5
; 1551          1612  5             IRAB[IRB$V_SKIP_NEXT] = 1;
; 1552          1613  5
; 1553          1614  5             ! First time call after $CONNECT or $REWIND. Then what we
; 1554          1615  5             ! want to retrieve is the very first record, so don't skip
; 1555          1616  5             ! next record.
; 1556          1617  5             !
; 1557          1618  5             IF .IRAB[IRB$L_CUR_VBN] EQL 0
; 1558          1619  5             THEN
; 1559          1620  5                 IRAB[IRB$V_SKIP_NEXT] = 0;
; 1560          1621  5
; 1561          1622  5             RETURN_ON_ERROR (RM$KEY_DESC(.IRAB[IRB$B_CUR_KREF]));
; 1562          1623  5
; 1563          1624  5             IRAB[IRB$B_KEYSZ] = .IDX_DFN[IDX$B_KEYSZ];
; 1564          1625  5             CH$MOVE(.IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(1), KEYBUF_ADDR(2));
; 1565          1626  4             END;
; 1566          1627  4
; 1567          1628  3         END;
; 1568          1629  3
; 1569          1630  3     [RAB$C_KEY] :
; 1570          1631  3 KEY :
; 1571          1632  4         BEGIN
; 1572          1633  4
; 1573          1634  4         LOCAL
; 1574          1635  4             KEYSIZE : BYTE,
; 1575          1636  4             KBF_ADDR : LONG;
; 1576          1637  4
; 1577          1638  4         IRAB[ IRB$V_CON_EOF ] = 0;
; 1578          1639  4
; 1579          1640  4         IRAB[IRB$V_SKIP_NEXT] = 0;
; 1580          1641  4
; 1581          1642  4         RETURN_ON_ERROR (RM$KEY_DESC(.RAB[RAB$B_KRF]));
; 1582          1643  4         KEYSIZE = .RAB[RAB$B_KSZ];
; 1583          1644  4
; 1584          1645  4         ! Check and setup for user key value.
; 1585          1646  4         !
; 1586          1647  4         IF .IDX_DFN[IDX$B_DATATYPE] EQL IDX$C_STRING
; 1587          1648  4           OR .IDX_DFN[IDX$B_SEGMENTS] GTR 1
; 1588          1649  4         THEN
; 1589          1650  5             BEGIN
; 1590          1651  5
; 1591          1652  7             IF ((.KEYSIZE EQL 0)
; 1592          1653  6                 OR
```

```
; 1593          1654  6                 (.KEYSIZE GTRU .IDX_DFN[IDX$B_KEYSZ]))
; 1594          1655  5             THEN
; 1595          1656  5                 RETURN RMSERR(KSZ);
; 1596          1657  5
; 1597          1658  5             END
; 1598          1659  4         ELSE
; 1599          1660  5             BEGIN
; 1600          1661  5
; 1601          1662  5             IF .KEYSIZE EQL 0
; 1602          1663  5             THEN
; 1603          1664  5                 KEYSIZE = .IDX_DFN[IDX$B_KEYSZ];
; 1604          1665  5
; 1605          1666  5             IF .KEYSIZE NEQU .IDX_DFN[IDX$B_KEYSZ]
; 1606          1667  5             THEN
; 1607          1668  5                 RETURN RMSERR(KSZ);
; 1608          1669  5
; 1609          1670  4             END;
; 1610          1671  4
; 1611          1672  4         IRAB[IRB$B_KEYSZ] = .KEYSIZE;
; 1612          1673  4         KBF_ADDR = .RAB [RAB$L_KBF];
; 1613        P 1674  4         IFNORD(KEYSIZE, .KBF_ADDR, IRAB[IRB$B_MODE],
; 1614          1675  4             RETURN RMSERR(KBF));
; 1615          1676  4
; 1616          1677  4         ! Move the user's key into keybuffer 2.
; 1617          1678  4         !
; 1618          1679  4         CH$MOVE(.KEYSIZE, .KBF_ADDR, KEYBUF_ADDR(2));
; 1619          1680  4
; 1620          1681  4         ! If key type is packed decimal then check it for valid nibbles.
; 1621          1682  4         !
; 1622          1683  4         IF .IDX_DFN[IDX$B_DATATYPE] EQLU IDX$C_PACKED
; 1623          1684  4         THEN
; 1624          1685  4             RETURN_ON_ERROR (RM$PCKDEC_CHECK());
; 1625          1686  4
; 1626          1687  4         ! Check that key match is logically consistent.
; 1627          1688  4         !
; 1628          1689  4         IF .RAB[RAB$V_KGE]
; 1629          1690  4         THEN
; 1630          1691  4
; 1631          1692  4             IF .RAB[RAB$V_KGT]
; 1632          1693  4             THEN
; 1633          1694  5                 RETURN RMSERR(POP)
; 1634          1695  4             ELSE
; 1635          1696  5                 BEGIN
; 1636          1697  5                 IRAB[IRB$V_SRCHGE] = 1;
; 1637          1698  5                 LEAVE KEY
; 1638          1699  5
; 1639          1700  5                 END
; 1640          1701  4         ELSE
; 1641          1702  4
; 1642          1703  4             IF .RAB[RAB$V_KGT]
; 1643          1704  4             THEN
; 1644          1705  5                 BEGIN
; 1645          1706  5                 IRAB[IRB$V_SRCHGT] = 1;
; 1646          1707  5                 LEAVE KEY
; 1647          1708  5
; 1648          1709  4                 END;
; 1649          1710  4
```

```
; 1650   1711  4          !+
; 1651   1712  4          ! At this point we have determined that this a random access for an
; 1652   1713  4          ! exact match by key.  Now try to find out if this is for the
; 1653   1714  4          ! current record, i.e., is it the same one we just got.  This will
; 1654   1715  4          ! be checked only for primary key. The following conditions must be
; 1655   1716  4          ! met to take this optimization:
; 1656   1717  4          !
; 1657   1718  4          !   Previous operation was a GET.
; 1658   1719  4          !   This operation is for primary key.
; 1659   1720  4          !   Duplicates aren't allowed on primary key.
; 1660   1721  4          !   The full key size is being used.
; 1661   1722  4          !   The key value matches the saved primary key value of the
; 1662   1723  4          !           current record (in keybuffer 3).
; 1663   1724  4          !   There is a current record (rp_vbn neq 0).
; 1664   1725  4          !   The current record is already locked, if locking required.
; 1665   1726  4          !   The new record is to be locked, if locking required.
; 1666   1727  4          !-
; 1667   1728  4          !
; 1668   1729  4          IF .IRAB[IRB$V_FIND_LAST]
; 1669   1730  4              OR
; 1670   1731  4              .IDX_DFN[IDX$B_KEYREF] NEQ 0
; 1671   1732  4              OR
; 1672   1733  4              .IDX_DFN[IDX$V_DUPKEYS]
; 1673   1734  4          THEN
; 1674   1735  4              LEAVE KEY;
; 1675   1736  4
; 1676   1737  4          IF .IRAB[IRB$B_KEYSZ] NEQ .IDX_DFN[IDX$B_KEYSZ]
; 1677   1738  4          THEN
; 1678   1739  4              LEAVE KEY;
; 1679   1740  4
; 1680   1741  5          BEGIN
; 1681   1742  5
; 1682   1743  5          LOCAL
; 1683   1744  5              SIZE;
; 1684   1745  5
; 1685   1746  5          SIZE = .IRAB[IRB$B_KEYSZ];
; 1686   1747  5
; 1687   1748  5          IF NOT CH$EQL(.SIZE, KEYBUF_ADDR(2), .SIZE, KEYBUF_ADDR(3))
; 1688   1749  5          THEN
; 1689   1750  5              LEAVE KEY;
; 1690   1751  5
; 1691   1752  4          END;
; 1692   1753  5          BEGIN
; 1693   1754  5
; 1694   1755  5          LOCAL
; 1695   1756  5              VBN;
; 1696   1757  5
; 1697   1758  5          IF (VBN = .IRAB[IRB$L_UDR_VBN]) EQL 0
; 1698   1759  5          THEN
; 1699   1760  5              LEAVE KEY;
; 1700   1761  5
; 1701   1762  5          ! If record locking is required, make sure this record is already
; 1702   1763  5          ! locked, otherwise it may be deleted or locked by another
; 1703   1764  5          ! accessor.  Also that the new record is to be locked also,
; 1704   1765  5          ! otherwise there is potentially an obscure window where it could
; 1705   1766  5          ! be deleted while reaccessing the bucket after the current lock is
; 1706   1767  5          ! released.
```

```
; 1707          1768  5             !
; 1708          1769  5             IF  NOT .IFAB[IFB$V_NORECLK]
; 1709          1770  5                  OR
; 1710          1771  6                 (.IFAB[IFB$V_RU_RLK]
; 1711          1772  6                          AND
; 1712          1773  6                         .IFAB[IFB$V_RUP])
; 1713          1774  5             THEN
; 1714          1775  6                 BEGIN
; 1715          1776  6
; 1716          1777  6                 LOCAL
; 1717          1778  6                     ST;
; 1718          1779  6
; 1719          1780  6                 IF .RAB[RAB$V_NLK]
; 1720          1781  6                     AND
; 1721          1782  6                    NOT .IFAB[IFB$V_RUP]
; 1722          1783  6                 THEN
; 1723          1784  6                     LEAVE KEY;
; 1724          1785  6
; 1725          1786  6                 ! If the user has requested waiting for record locking,
; 1726          1787  6                 ! disable such waiting at this time. RMS does not want to
; 1727          1788  6                 ! wait because it is just interested in whether this stream
; 1728          1789  6                 ! has locked the record or there is a window. RMS does not
; 1729          1790  6                 ! want to wait for it here if it has to. The lock logic later
; 1730          1791  6                 ! on will do the waiting if it is necessary.
; 1731          1792  6                 !
; 1732          1793  6                 IF .RAB[RAB$V_WAT]
; 1733          1794  6                 THEN
; 1734          1795  6                     IRAB[IRB$V_NO_Q_WAIT] = 1;
; 1735          1796  6
; 1736          1797  6                 ST = RM$QUERY_LCK(.VBN, .IRAB[IRB$W_UDR_ID]);
; 1737          1798  6
; 1738          1799  7                 IF .ST<0, 16> NEQU RMSSUC(OK_ALK)
; 1739          1800  6                 THEN
; 1740          1801  6                     LEAVE KEY
; 1741          1802  6
; 1742          1803  5                 END;
; 1743          1804  5
; 1744          1805  4             END;
; 1745          1806  4
; 1746          1807  4             ! If we are here we have determined that this is the same record
; 1747          1808  4             ! that we already have locked as the current record. Flag that
; 1748          1809  4             ! state by setting DUP. By leaving the unlock block, the current
; 1749          1810  4             ! record is not unlocked.
; 1750          1811  4             !
; 1751          1812  4             !                   >>>=====> NOTE <=====<<<
; 1752          1813  4             !
; 1753          1814  4             ! Setting DUP will cause us to fall into an optimization below,
; 1754          1815  4             ! which will change our access to sequential in order to quickly
; 1755          1816  4             ! reaccess the record that we have been waiting on (and should
; 1756          1817  4             ! now have locked). There are some problems with this way of
; 1757          1818  4             ! thinking, especially when the following are true:
; 1758          1819  4             !
; 1759          1820  4             !   - we just had this record, and released it (which means
; 1760          1821  4             !           that our record context is STILL INTACT)
; 1761          1822  4             !   - the record we were waiting on was deleted after we
; 1762          1823  4             !           released it
; 1763          1824  4             !   - keyed access
```

```
; 1764          1825  4              !   - exact match by primary key
; 1765          1826  4              !   - no dups
; 1766          1827  4              !
; 1767          1828  4              ! The above conditions will give us the first non-deleted record,
; 1768          1829  4              ! regardless of key (since we temporarily used sequential access).
; 1769          1830  4              ! And since KEYBUFFER 3 still has the key from when we last had
; 1770          1831  4              ! this record (before we released it), it will match our search
; 1771          1832  4              ! key.  I have attempted in RM$GET3B to prevent coming back here a
; 1772          1833  4              ! second time if these conditions are true.
; 1773          1834  4
; 1774          1835  4              IRAB[IRB$V_DUP] = 1;
; 1775          1836  4              LEAVE UNLOCK;
; 1776          1837  4
; 1777          1838  3              END;                                ! of block KEY
; 1778          1839  3
; 1779          1840  3          [RAB$C_RFA] :
; 1780          1841  3
; 1781          1842  3              ! RFA access -- check RFA for legality and setup for primary key
; 1782          1843  3              !           access for next record pointer (NRP) data.
; 1783          1844  3              !
; 1784          1845  4              BEGIN
; 1785          1846  4
; 1786          1847  4              IRAB[ IRB$V_CON_EOF ] = 0;
; 1787          1848  4
; 1788          1849  4              IRAB[IRB$V_SKIP_NEXT] = 0;          ! flag random access
; 1789          1850  4
; 1790          1851  4              IF .RAB[RAB$L_RFA0] EQL 0
; 1791          1852  4                  OR
; 1792          1853  4                  .RAB[RAB$W_RFA4] EQL 0
; 1793          1854  4              THEN
; 1794          1855  4                  RETURN RMSERR(RFA);
; 1795          1856  4
; 1796          1857  4              RETURN_ON_ERROR (RM$KEY_DESC(0));
; 1797          1858  4
; 1798          1859  3              END;
; 1799          1860  3
; 1800          1861  3          [OUTRANGE] :
; 1801          1862  3              RETURN RMSERR(RAC);
; 1802          1863  3
; 1803          1864  3          TES;
; 1804          1865  3
; 1805          1866  3      ! The current record is now unlocked before accessing the new record,
; 1806          1867  3      ! unless it has already been determined that the new record is the same as
; 1807          1868  3      ! the old current record, in which case this block was left and this code
; 1808          1869  3      ! is skipped.
; 1809          1870  3      !
; 1810          1871  3      IF TESTBITSC(IRAB[IRB$V_UNLOCK_RP])
; 1811          1872  3      THEN
; 1812          1873  3
; 1813          1874  3          ! If RMS is performing a re-positioning then unlock the record
; 1814          1875  3          ! positioned to during the previous positioning attempt; otherwise,
; 1815          1876  3          ! unlock the current record (locked during the previous positioning
; 1816          1877  3          ! operation) if there is one.
; 1817          1878  3          !
; 1818          1879  3          IF .REPOS_STATUS
; 1819          1880  3          THEN
; 1820          1881  3              RM$UNLOCK (.IRAB[IRB$L_NEXT_VBN], .IRAB[IRB$W_NEXT_ID])
```

```
; 1821          1882  3         ELSE
; 1822          1883  3             IF .IRAB[IRB$L_UDR_VBN] NEQU 0
; 1823          1884  3             THEN
; 1824          1885  3                 RM$UNLOCK (.IRAB[IRB$L_UDR_VBN], .IRAB[IRB$W_UDR_ID]);
; 1825          1886  3
; 1826          1887  2     END;                                        ! of block UNLOCK
; 1827          1888  2     IRAB[IRB$B_RP_KREF] = .IDX_DFN[IDX$B_KEYREF];
; 1828          1889  2
; 1829          1890  2     ! Get record block 2 -- position and perform lock logic for record which
; 1830          1891  2     ! the IRAB search context data describes.
; 1831          1892  2     !
; 1832          1893  3     BEGIN
; 1833          1894  3
; 1834          1895  3     LOCAL
; 1835          1896  3         STATUS;
; 1836          1897  3
; 1837          1898  4     IF NOT (STATUS =
; 1838          1899  5         BEGIN
; 1839          1900  5
; 1840          1901  5         LOCAL
; 1841          1902  5             RAC : BYTE;
; 1842          1903  5
; 1843          1904  5         RAC = .RAB[RAB$B_RAC];
; 1844          1905  5
; 1845          1906  5         IF .IRAB[IRB$V_DUP]                     ! re-accessing current record
; 1846          1907  5         THEN
; 1847          1908  6             BEGIN
; 1848          1909  6
; 1849          1910  6             ! If next record info also for primary key, then use sequential
; 1850          1911  6             ! postioning code and nrp info - it's faster. irb$v_skip_next
; 1851          1912  6             ! will be clear in this case so that record itself is retrieved.
; 1852          1913  6             !
; 1853          1914  6             ASSUME_C(RAB$C_SEQ, 0);
; 1854          1915  6
; 1855          1916  6             ! sneaky way to set rac = rab$c_seq
; 1856          1917  6             !
; 1857          1918  6             RAC = .IRAB[IRB$B_CUR_KREF];
; 1858          1919  6
; 1859          1920  6             IF .RAC NEQ 0                       ! when cur_kref = 0.
; 1860          1921  6             THEN
; 1861          1922  6                 RAC = RAB$C_RFA;
; 1862          1923  6
; 1863          1924  5             END;
; 1864          1925  5
; 1865          1926  5         CASE .RAC FROM RAB$C_SEQ TO RAB$C_RFA OF
; 1866          1927  5             SET
; 1867          1928  5
; 1868          1929  5             ! Sequential access.
; 1869          1930  5             !
; 1870          1931  5             [RAB$C_SEQ] : STATUS = RM$POS_SEQ();
; 1871          1932  5
; 1872          1933  5             ! Random access by key.
; 1873          1934  5             !
; 1874          1935  5             [RAB$C_KEY] : STATUS = RM$POS_KEY();
; 1875          1936  5
; 1876          1937  5             ! Random access by RFA.
; 1877          1938  5             !
```

```
; 1878          1939  5                [RAB$C_RFA] : STATUS = RM$POS_RFA();
; 1879          1940  5
; 1880          1941  5                TES
; 1881          1942  5
; 1882          1943  4            END)
; 1883          1944  3        THEN
; 1884          1945  3            RETURN .STATUS;
; 1885          1946  3
; 1886          1947  3        ! Setup record pointer (RP) to the RFA/RRV of retrieved record.
; 1887          1948  3        !
; 1888          1949  3        IRAB[IRB$W_NEXT_ID] = RM$RECORD_ID();
; 1889          1950  3
; 1890          1951  3        AP = 3;
; 1891          1952  3
; 1892          1953  4        BEGIN
; 1893          1954  4
; 1894          1955  4        GLOBAL REGISTER
; 1895          1956  4            R_BDB;
; 1896          1957  4
; 1897          1958  4        IRAB[IRB$L_NEXT_VBN] = RM$RECORD_VBN();
; 1898          1959  3        END;
; 1899          1960  3
; 1900          1961  3        ! Move the key of the internally current record into keybuffer 2 in
; 1901          1962  3        ! preparation for updating the local NRP context. It will not be necessary
; 1902          1963  3        ! to extract the key, if positioning was done by means of an alternate key
; 1903          1964  3        ! index, because as part of that positioning, the key would have been moved
; 1904          1965  3        ! into keybuffer 2.
; 1905          1966  3        !
; 1906          1967  3        IF .IRAB[IRB$B_RP_KREF] EQLU 0
; 1907          1968  3        THEN
; 1908          1969  3
; 1909          1970  3            ! If any of the following conditions holds, the key of the internally
; 1910          1971  3            ! current record (which must be a primary data record) must be
; 1911          1972  3            ! extracted from the record itself.
; 1912          1973  3            !
; 1913          1974  3            ! 1. If the file is a prologue 1 or 2 file.
; 1914          1975  3            ! 2. If the record was retrieved randomly.
; 1915          1976  3            ! 3. If key compression is not enabled in this key of reference.
; 1916          1977  3            ! 4. If deleted records were encountered during the positioning.
; 1917          1978  3            !
; 1918          1979  5            IF ((.IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
; 1919          1980  4                OR
; 1920          1981  5                (.RAB[RAB$B_RAC] NEQU RAB$C_SEQ)
; 1921          1982  4                OR
; 1922          1983  4                NOT .IDX_DFN[IDX$V_KEY_COMPR]
; 1923          1984  4                OR
; 1924          1985  4                .IRAB[IRB$V_DEL_SEEN])
; 1925          1986  3            THEN
; 1926          1987  4                BEGIN
; 1927          1988  4
; 1928          1989  4                GLOBAL REGISTER
; 1929          1990  4                    R_BDB;
; 1930          1991  4
; 1931          1992  4                AP = 0;
; 1932          1993  4                RM$RECORD_KEY (KEYBUF_ADDR(2));
; 1933          1994  4                END
; 1934          1995  4
```

```
; 1935          1996  4          ! If this is a prologue 3 file, and RMS is positioning sequentially,
; 1936          1997  4          ! then RMS may use the key of the last retrieved record, stored in
; 1937          1998  4          ! keybuffer 1 to supply any characters front compressed off the current
; 1938          1999  4          ! key provided no intervening records were encountered between the
; 1939          2000  4          ! last retrieved, and the new record.
; 1940          2001  4          !
; 1941          2002  3          ELSE
; 1942          2003  4              BEGIN
; 1943          2004  4
; 1944          2005  4              LOCAL
; 1945          2006  4                  KEY     : REF BBLOCK;
; 1946          2007  4
; 1947          2008  4              MACRO
; 1948          2009  4                  KEY_LEN = 0,0,8,0 %,
; 1949          2010  4                  CMP_CNT = 1,0,8,0 %;
; 1950          2011  4
; 1951          2012  4              KEY = .REC_ADDR + RM$REC_OVHD(0);
; 1952          2013  4
; 1953          2014  4              CH$COPY (.KEY[CMP_CNT], KEYBUF_ADDR(1),
; 1954          2015  4                       .KEY[KEY_LEN], .KEY+2,
; 1955          2016  4                       .(.KEY + .KEY[KEY_LEN] + 1),
; 1956          2017  4                       .IDX_DFN[IDX$B_KEYSZ], KEYBUF_ADDR(2));
; 1957          2018  3              END;
; 1958          2019  3
; 1959          2020  3      ! Don't do any record-locking if there aren't any writers of the file, and
; 1960          2021  3      ! if pseudo record locking is not to be done.
; 1961          2022  3      !
; 1962          2023  3      IF  .IFAB[IFB$V_NORECLK]
; 1963          2024  3          AND
; 1964          2025  4          NOT (.IFAB[IFB$V_RU_RLK]
; 1965          2026  4                AND
; 1966          2027  4                .IFAB[IFB$V_RUP])
; 1967          2028  3      THEN
; 1968          2029  3          RETURN .STATUS;
; 1969          2030  3
; 1970          2031  2      END;                                  ! of block defining local STATUS
; 1971          2032  2
; 1972          2033  3      BEGIN
; 1973          2034  3
; 1974          2035  3      LOCAL
; 1975          2036  3          STATUS;
; 1976          2037  3
; 1977          2038  4      BEGIN
; 1978          2039  4
; 1979          2040  4      LABEL
; 1980          2041  4          OK_WAT;
; 1981          2042  4
; 1982          2043  4      ! Flag no special action needed for unlocking the RP
; 1983          2044  4      !
; 1984          2045  4      AP = 0;
; 1985          2046  4
; 1986          2047  4      !
; 1987          2048  4      ! only query_lock the record if:
; 1988          2049  4      !
; 1989          2050  4      !   1. The user has specified no locking ( NLK ) and is not in a Recovery
; 1990          2051  4      !      Unit.
; 1991          2052  4      !
```

```
; 1992          2053  4      !   2. The file is opened for read ( not WRTACC )
; 1993          2054  4      !      and the user has not specified read-only locking ( not REA )
; 1994          2055  4      !
; 1995          2056  5      IF  (.RAB[RAB$V_NLK]
; 1996          2057  5                AND
; 1997          2058  5               NOT .IFAB[IFB$V_RUP])
; 1998          2059  5
; 1999          2060  4          OR
; 2000          2061  6          ( (NOT .IFAB[IFB$V_WRTACC])
; 2001          2062  5             AND
; 2002          2063  5            (NOT .RAB[RAB$V_REA]) )
; 2003          2064  4      THEN
; 2004          2065  4          STATUS = RM$QUERY_LCK (.IRAB[IRB$L_NEXT_VBN], .IRAB[IRB$W_NEXT_ID])
; 2005          2066  4      ELSE
; 2006          2067  4          STATUS = RM$LOCK (.IRAB[IRB$L_NEXT_VBN], .IRAB[IRB$W_NEXT_ID]);
; 2007          2068  4
; 2008          2069  4
; 2009          2070  4      ! OK_WAT success status means we had to wait for someone else to unlock the
; 2010          2071  4      ! record.  To wait, we deaccessed the bucket.  Therefore, we must reaccess
; 2011          2072  4      ! it, and we can  use the record pointer information for this. Deaccessing
; 2012          2073  4      ! the bucket also means that our NRP context updating information in the
; 2013          2074  4      ! IRAB cannot longer be considered to be valid.
; 2014          2075  4      !
; 2015          2076  5      IF .STATUS EQL RMSSUC(OK_WAT)
; 2016          2077  4      THEN
; 2017          2078  4 OK_WAT:
; 2018          2079  5          BEGIN
; 2019          2080  5
; 2020          2081  5          LOCAL
; 2021          2082  5              TEMP_STATUS;
; 2022          2083  5
; 2023          2084  5          ! Reposition to the record using record pointer contents. If it is
; 2024          2085  5          ! possible that some reclamation maybe done make sure the primary data
; 2025          2086  5          ! bucket is exclusively accessed.
; 2026          2087  5          !
; 2027          2088  5          IF .IFAB[IFB$V_WRTACC]
; 2028          2089  5              AND
; 2029          2090  5             .IFAB[IFB$V_RU]
; 2030          2091  5          THEN
; 2031          2092  5              IRAB[IRB$B_CACHEFLGS] = CSH$M_LOCK;
; 2032          2093  5
; 2033          2094  6          IF NOT (TEMP_STATUS = RM$FIND_BY_RRV (.IRAB[IRB$L_NEXT_VBN],
; 2034          2095  6                                               .IRAB[IRB$W_NEXT_ID],
; 2035          2096  6                                               0))
; 2036          2097  5          THEN
; 2037          2098  5              STATUS = .TEMP_STATUS;
; 2038          2099  5
; 2039          2100  5          ! If RMS after re-positioning to the record finds that it had been
; 2040          2101  5          ! deleted within a Recovery Unit, then RMS will have to re-position
; 2041          2102  5          ! after deleting the record for good if it has write access to the
; 2042          2103  5          ! file.
; 2043          2104  5          !
; 2044          2105  5          IF .STATUS
; 2045          2106  5              AND
; 2046          2107  5             .REC_ADDR[IRC$V_RU_DELETE]
; 2047          2108  5          THEN
; 2048          2109  5              LEAVE OK_WAT;
```

```
; 2049          2110  5
; 2050          2111  5          IF .STATUS
; 2051          2112  5          THEN
; 2052          2113  5
; 2053          2114  5              ! If our key of reference is the primary key, then we can reclaim
; 2054          2115  5              ! our NRP updating information from the primary data bucket's
; 2055          2116  5              ! VBN and the record ID.
; 2056          2117  5              !
; 2057          2118  6              IF (.IDX_DFN[IDX$B_KEYREF] EQL 0)
; 2058          2119  5              THEN
; 2059          2120  6                  BEGIN
; 2060          2121  6                  IRAB[IRB$L_RFA_VBN] = .BBLOCK[.IRAB[IRB$L_CURBDB], BDB$L_VBN];
; 2061          2122  6                  IRAB[IRB$W_RFA_ID] =  IRC$_ID(REC_ADDR);
; 2062          2123  6                  END
; 2063          2124  6
; 2064          2125  6              ! If the key of reference is not the primary key, then RMS has no
; 2065          2126  6              ! easy way to reclaim the NRP list updating information which is
; 2066          2127  6              ! for the SIDR bucket (long since released) and not the primary
; 2067          2128  6              ! data bucket. Since the stream which has the record locked might
; 2068          2129  6              ! delete the SIDR array positioned to but not the primary data
; 2069          2130  6              ! record itself (by means of an $UPDATE), RMS must re-position
; 2070          2131  6              ! in order to guarentee that the key of the SIDR array it
; 2071          2132  6              ! positions to is actually represented in the primary data record
; 2072          2133  6              ! to be returned. The alternate success status, and the fact that
; 2073          2134  6              ! the key of reference is other than the primary will force another
; 2074          2135  6              ! attempt to access the primary data bucket after accessing the
; 2075          2136  6              ! necessary SIDR, and to lock the next record.
; 2076          2137  6              !
; 2077          2138  5              ELSE
; 2078          2139  6                  BEGIN
; 2079          2140  6
; 2080          2141  6                  GLOBAL REGISTER
; 2081          2142  6                      R_BDB_STR;
; 2082          2143  6
; 2083          2144  6                  RELEASE (IRAB[IRB$L_CURBDB]);
; 2084          2145  6                  END
; 2085          2146  5          ELSE
; 2086          2147  5              IRAB[IRB$L_CURBDB] = 0;
; 2087          2148  5
; 2088          2149  4          END;
; 2089          2150  4
; 2090          2151  4      ! If RMS finds that the current record has been modified within a Recovery
; 2091          2152  4      ! Unit, then subject it to further processing before deciding whether to
; 2092          2153  4      ! return it as the non-deleted primary data record to be returned to the
; 2093          2154  4      ! user, or whether to return a status to force RMS to re-position.
; 2094          2155  4      !
; 2095          2156  4      IF  .STATUS
; 2096          2157  4           AND
; 2097          2158  4          .IRAB[IRB$L_CURBDB] NEQU 0
; 2098          2159  4           AND
; 2099          2160  5          (.REC_ADDR[IRC$V_RU_DELETE]
; 2100          2161  5                  OR
; 2101          2162  5                  .REC_ADDR[IRC$V_RU_UPDATE])
; 2102          2163  4      THEN
; 2103          2164  5          BEGIN
; 2104          2165  5
; 2105          2166  5          LOCAL
```

```
; 2106          2167  5            RECORD_ID : WORD;
; 2107          2168  5
; 2108          2169  5        RECORD_ID = .REC_ADDR[IRC$W_ID];
; 2109          2170  5
; 2110          2171  5        ! If the file has been open for write access, then attempt to delete
; 2111          2172  5        ! the record if it was deleted within a Recovery Unit, or re-format
; 2112          2173  5        ! the record if it was updated within a Recovery Unit.
; 2113          2174  5        !
; 2114          2175  5        IF .IFAB[IFB$V_WRTACC]
; 2115          2176  5        THEN
; 2116          2177  5            RM$RU_RECLAIM();
; 2117          2178  5
; 2118          2179  5        ! If the record had been deleted within a Recovery Unit, then RMS will
; 2119          2180  5        ! not return this record to the user as a non-deleted primary data
; 2120          2181  5        ! record. Therefore, release the primary data bucket, and change the
; 2121          2182  5        ! return status to 0 if RMS did not have to wait for the record lock, or
; 2122          2183  5        ! change the return status to RMS$_DEL if RMS had to wait for the
; 2123          2184  5        ! record lock. Returning a status of RMS$_DEL in the latter case will
; 2124          2185  5        ! allow the information that RMS had to wait for a record lock to
; 2125          2186  5        ! eventually be returned to the user along with the non-deleted primary
; 2126          2187  5        ! data record when such a record is eventually found.
; 2127          2188  5        !
; 2128          2189  5        IF .RECORD_ID NEQU .REC_ADDR[IRC$W_ID]
; 2129          2190  5            OR
; 2130          2191  5           .REC_ADDR[IRC$V_RU_DELETE]
; 2131          2192  5        THEN
; 2132          2193  6            BEGIN
; 2133          2194  6
; 2134          2195  6            GLOBAL REGISTER
; 2135          2196  6                R_BDB_STR;
; 2136          2197  6
; 2137          2198  6            RELEASE (IRAB[IRB$L_CURBDB]);
; 2138          2199  6
; 2139          2200  7            IF .STATUS<0,16> EQLU RMSSUC(OK_WAT)
; 2140          2201  6            THEN
; 2141          2202  7                STATUS = RMSERR(DEL)
; 2142          2203  6            ELSE
; 2143          2204  6                STATUS = 0;
; 2144          2205  5            END;
; 2145          2206  4        END;
; 2146          2207  4
; 2147          2208  4     ! Return here if QUERY_LCK.
; 2148          2209  4     !
; 2149          2210  5     IF  (.RAB[RAB$V_NLK]
; 2150          2211  5                 AND
; 2151          2212  5                NOT .IFAB[IFB$V_RUP])
; 2152          2213  4          OR
; 2153          2214  5        (NOT .IFAB[IFB$V_WRTACC]
; 2154          2215  5                 AND
; 2155          2216  5                NOT .RAB[RAB$V_REA])
; 2156          2217  4     THEN
; 2157          2218  4         RETURN .STATUS;
; 2158          2219  4
; 2159          2220  3     END;
; 2160          2221  3
; 2161          2222  3     ! If UNLOCK_RP was set coming here, it can only mean that this was a
; 2162          2223  3     ! reaccessing of a previously automatically locked record that was not
```

```
; 2163    2224  3     ! unlocked at the beginning of this operation to avoid a locking window.
; 2164    2225  3     ! It will get an ok_alk status (not suc) from rm$lock.  It wants to release
; 2165    2226  3     ! the current record on potential buffer errors.  The case where we don't
; 2166    2227  3     ! want to release the now current record lock is if the status from rm$lock
; 2167    2228  3     ! was rms$_ok_alk (i.e., not suc) which meant that it had been previously
; 2168    2229  3     ! manually locked, and should remain that way even if this operation fails.
; 2169    2230  3     !
; 2170    2231  3     IF TESTBITCC(IRAB[IRB$V_UNLOCK_RP])
; 2171    2232  3     THEN
; 2172    2233  4         IF  .STATUS<0,16> EQLU RMSSUC()
; 2173    2234  3              OR
; 2174    2235  4             .STATUS<0,16> EQLU RMSSUC(OK_WAT)
; 2175    2236  3              OR
; 2176    2237  4             .STATUS<0,16> EQLU RMSSUC(OK_RULK)
; 2177    2238  3         THEN
; 2178    2239  3             AP = 1
; 2179    2240  3         ELSE
; 2180    2241  3
; 2181    2242  3             ! If it is necessary for us to release the record lock set
; 2182    2243  3             ! IRB$V_UNLOCK_RP. It will only be necessary to release the record
; 2183    2244  3             ! lock in some circumstances when we have had to stall waiting for
; 2184    2245  3             ! it, and whenever RMS has managed to position to a record that was
; 2185    2246  3             ! deleted within a Recovery Unit. In the former case if after
; 2186    2247  3             ! waiting for the lock, we have some problem reaccessing the bucket,
; 2187    2248  3             ! or we find that the record is deleted while we were waiting, then
; 2188    2249  3             ! we must release the record lock. If we are positioning by means
; 2189    2250  3             ! of an alternate key and we have had to stall waiting for the
; 2190    2251  3             ! record lock, and this is an operation where the NRP list must be
; 2191    2252  3             ! updated (any operation but a nonrandom $FIND), then the record
; 2192    2253  3             ! lock must also be released.
; 2193    2254  3             !
; 2194    2255  4             BEGIN
; 2195    2256  4
; 2196    2257  5             IF  (.IRAB[IRB$L_CURBDB] EQL 0)
; 2197    2258  4              AND
; 2198    2259  5             (.STATUS<0,16> NEQ RMSERR(RLK))
; 2199    2260  4             THEN
; 2200    2261  4                 IRAB[IRB$V_UNLOCK_RP] = 1;
; 2201    2262  4
; 2202    2263  4             RETURN .STATUS;
; 2203    2264  4             END
; 2204    2265  3     ELSE
; 2205    2266  3         AP = 1;
; 2206    2267  3
; 2207    2268  3     IF NOT .RAB[RAB$V_ULK]
; 2208    2269  3     THEN
; 2209    2270  3         IRAB[IRB$V_UNLOCK_RP] = 1;
; 2210    2271  3
; 2211    2272  3     RETURN .STATUS;
; 2212    2273  3
; 2213    2274  3     END                                 ! of local block defining STATUS
; 2214    2275  1     END;                                ! of routine
```

```
                5E          18  C2 00000 GET_RECORD:
                                         SUBL2   #24, SP                          ; 1427
             40    A9  94 00003          CLRB    64(IRAB)                         ; 1507
             42    A9  B4 00006          CLRW    66(IRAB)                         ; 1508
        14   AE    04    A9  9E 00009    MOVAB   4(IRAB), 20(SP)                  ; 1509
        14   BE  1000    8F  AA 0000E    BICW2   #4096, @20(SP)                   ;
             38    A8  D4 00014          CLRL    56(RAB)                          ; 1515
   02        00    1E    A8  8F 00017    CASEB   30(RAB), #0, #2                  ; 1524
 01A3      0097      000D      0001C 1$:  .WORD   2$-1$,-                          ;
                                                 12$-1$,-                         ;
                                                 29$-1$                           ;
        50  8644    8F  3C 00022          MOVZWL  #34372, R0                       ; 1862
             4C  11 00027          BRB     8$                                       ;
   42   14   BE    17  E0 00029 2$:       BBS     #23, @20(SP), 7$                 ; 1549
   38   14   BE    05  E1 0002E           BBC     #5, @20(SP), 6$                  ; 1553
   33   14   BE    09  E0 00033           BBS     #9, @20(SP), 6$                  ; 1555
   0C   14   BE    0B  E5 00038           BBCC    #11, @20(SP), 3$                 ; 1563
   2E   14   BE    01  E0 0003D           BBS     #1, @20(SP), 7$                  ; 1570
        7E  00C3    C9  9A 00042          MOVZBL  195(IRAB), -(SP)                 ; 1574
             08  11 00047          BRB     4$                                       ;
           FC87  30 00049 3$:      BSBW    SETUP_NRP_DATA                           ; 1584
        7E  00C2    C9  9A 0004C          MOVZBL  194(IRAB), -(SP)                 ; 1585
          0000G  30 00051 4$:      BSBW    RM$KEY_DESC                              ;
        5E          04  C0 00054          ADDL2   #4, SP                           ;
   00A6 C9    20    A7  90 00057          MOVB    32(IDX_DFN), 166(IRAB)           ; 1588
   06   06   A8    04  E1 0005D           BBC     #4, 6(RAB), 5$                   ; 1596
   48 00A2   CA    02  E1 00062           BBC     #2, 162(IFAB), 11$               ; 1598
           01A0  31 00068 5$:      BRW     36$                                      ; 1600
   08   14   BE    01  E1 0006B 6$:       BBC     #1, @20(SP), 9$                  ; 1608
        50  827A    8F  3C 00070 7$:      MOVZWL  #33402, R0                       ; 1610
           03B6  31 00075 8$:      BRW     71$                                      ;
        14   BE  0800    8F  A8 00078 9$:       BISW2   #2048, @20(SP)           ; 1612
             00A8    C9  D5 0007E          TSTL    168(IRAB)                        ; 1618
             06  12 00082          BNEQ    10$                                      ;
        14   BE  0800    8F  AA 00084          BICW2   #2048, @20(SP)           ; 1620
        7E  00C3    C9  9A 0008A 10$:      MOVZBL  195(IRAB), -(SP)                 ; 1622
          0000G  30 0008F          BSBW    RM$KEY_DESC                              ;
        5E          04  C0 00092          ADDL2   #4, SP                           ;
        DD          50  E9 00095          BLBC    STATUS, 8$                       ;
   00A6 C9    20    A7  90 00098          MOVB    32(IDX_DFN), 166(IRAB)           ; 1624
        51    20    A7  9A 0009E          MOVZBL  32(IDX_DFN), R1                  ; 1625
        50  00B4    CA  3C 000A2          MOVZWL  180(IFAB), R0                    ;
        50    60    A9  C0 000A7          ADDL2   96(IRAB), R0                     ;
   60   60   B9    51  28 000AB          MOVC3   R1, @96(IRAB), (R0)              ;
           0135  31 000B0 11$:      BRW     33$                                      ; 1524
14 BE   01  17    00  F0 000B3 12$:      INSV    #0, #23, #1, @20(SP)             ; 1638
        14   BE  0800    8F  AA 000B9          BICW2   #2048, @20(SP)           ; 1640
        7E    35    A8  9A 000BF          MOVZBL  53(RAB), -(SP)                   ; 1642
          0000G  30 000C3          BSBW    RM$KEY_DESC                              ;
        5E          04  C0 000C6          ADDL2   #4, SP                           ;
        A9          50  E9 000C9          BLBC    STATUS, 8$                       ;
        50    34    A8  90 000CC          MOVB    52(RAB), KEYSIZE                 ; 1643
              1D    A7  95 000D0          TSTB    29(IDX_DFN)                      ; 1647
             06  13 000D3          BEQL    13$                                      ;
        01    1E    A7  91 000D5          CMPB    30(IDX_DFN), #1                  ; 1648
             0C  1B 000D9          BLEQU   14$                                      ;
             50  95 000DB 13$:      TSTB    KEYSIZE                          ; 1652
```

```
                        16  13 000DD          BEQL    16$                                  ;
          20   A7       50  91 000DF          CMPB    KEYSIZE, 32(IDX_DFN)                 ; 1654
                        17  1B 000E3          BLEQU   17$                                  ;
                        0E  11 000E5          BRB     16$                                  ; 1656
                        50  95 000E7 14$:     TSTB    KEYSIZE                              ; 1662
                        04  12 000E9          BNEQ    15$                                  ;
     50        20       A7  90 000EB          MOVB    32(IDX_DFN), KEYSIZE                 ; 1664
          20   A7       50  91 000EF 15$:     CMPB    KEYSIZE, 32(IDX_DFN)                 ; 1666
                        07  13 000F3          BEQL    17$                                  ;
     50       85A4      8F  3C 000F5 16$:     MOVZWL  #34212, R0                           ; 1668
                        44  11 000FA          BRB     20$                                  ;
    00A6   C9           50  90 000FC 17$:     MOVB    KEYSIZE, 166(IRAB)                   ; 1672
     52        30       A8  D0 00101          MOVL    48(RAB), KBF_ADDR                    ; 1673
62   50        0A       A9  0C 00105          PROBER  10(IRAB), KEYSIZE, (KBF_ADDR)        ; 1675
                        07  12 00110          BNEQ    18$                                  ;
     50       858C      8F  3C 00112          MOVZWL  #34188, R0                           ;
                        2D  11 00117          BRB     20$                                  ;
     51                 50  9A 00119 18$:     MOVZBL  KEYSIZE, R1                          ; 1679
     50       00B4      CA  3C 00116          MOVZWL  180(IFAB), R0                        ;
     50        60       A9  C0 0011B          ADDL2   96(IRAB), R0                         ;
60   62                 51  28 0011F          MOVC3   R1, (KBF_ADDR), (R0)                 ;
     05        1D       A7  91 00123          CMPB    29(IDX_DFN), #5                      ; 1683
                        06  12 00127          BNEQ    19$                                  ;
                      0000G 30 00129          BSBW    RM$PCKDEC_CHECK                      ; 1685
     11                 50  E9 0012C          BLBC    STATUS, 20$                          ;
     54        04       A8  9E 0012F 19$:     MOVAB   4(RAB), R4                           ; 1689
12   64                 15  E1 00133          BBC     #21, (R4), 22$                       ;
08   64                 16  E1 00137          BBC     #22, (R4), 21$                       ; 1692
     50       867C      8F  3C 0013B          MOVZWL  #34428, R0                           ; 1694
                      02EB  31 00140 20$:     BRW     71$                                  ;
     42   A9            10  88 00143 21$:     BISB2   #16, 66(IRAB)                        ; 1697
                        08  11 00147          BRB     23$                                  ; 1698
07   64                 16  E1 00149 22$:     BBC     #22, (R4), 24$                       ; 1703
     42   A9            02  88 0014D          BISB2   #2, 66(IRAB)                         ; 1706
                      0094  31 00151 23$:     BRW     33$                                  ; 1707
F8   04   A9            05  E0 00154 24$:     BBS     #5, 4(IRAB), 23$                     ; 1729
                  21    A7  95 00159          TSTB    33(IDX_DFN)                          ; 1731
                        F3  12 0015C          BNEQ    23$                                  ;
     EF        1C       A7  E8 0015E          BLBS    28(IDX_DFN), 23$                     ; 1733
     20   A7    00A6    C9  91 00162          CMPB    166(IRAB), 32(IDX_DFN)               ; 1737
                        7E  12 00168          BNEQ    33$                                  ;
     51       00A6      C9  9A 0016A          MOVZBL  166(IRAB), SIZE                      ; 1746
     50       00B4      CA  3C 0016F          MOVZWL  180(IFAB), R0                        ; 1748
                  60 B940  3F 00174          PUSHAW  @96(IRAB)[R0]                        ;
9E   60 B940            51  29 00178          CMPC3   SIZE, @96(IRAB)[R0], @(SP)+          ;
                        68  12 0017E          BNEQ    33$                                  ;
     51       00B0      C9  D0 00180          MOVL    176(IRAB), VBN                       ; 1758
                        61  13 00185          BEQL    33$                                  ;
0C        06   AA       03  E1 00187          BBC     #3, 6(IFAB), 25$                     ; 1769
27      00A2   CA       03  E1 0018C          BBC     #3, 162(IFAB), 28$                   ; 1771
21      00A2   CA       02  E1 00192          BBC     #2, 162(IFAB), 28$                   ; 1773
06   64                 14  E1 00198 25$:     BBC     #20, (R4), 26$                       ; 1780
46      00A2   CA       02  E1 0019C          BBC     #2, 162(IFAB), 33$                   ; 1782
04   64                 11  E1 001A2 26$:     BBC     #17, (R4), 27$                       ; 1793
     07   A9            01  88 001A6          BISB2   #1, 7(IRAB)                          ; 1795
     52       00BC      C9  3C 001AA 27$:     MOVZWL  188(IRAB), R2                        ; 1797
                      0000G 30 001AF          BSBW    RM$QUERY_LCK                         ;
```

```
                8039  8F          50  B1 001B2          CMPW    ST, #32825                          ; 1799
                                  2F  12 001B7          BNEQ    33$                                 ;
                  05  A9          10  88 001B9 28$:     BISB2   #16, 5(IRAB)                        ; 1835
                                  4C  11 001BD          BRB     36$                                 ; 1836
14  BE          01  17          00  F0 001BF 29$:     INSV    #0, #23, #1, @20(SP)                ; 1847
                  14  BE    0800  8F  AA 001C5          BICW2   #2048, @20(SP)                      ; 1849
                            10  A8  D5 001CB          TSTL    16(RAB)                             ; 1851
                                05  13 001CE          BEQL    30$                                 ;
                            14  A8  B5 001D0          TSTW    20(RAB)                             ; 1853
                                08  12 001D3          BNEQ    32$                                 ;
                    50    865C  8F  3C 001D5 30$:     MOVZWL  #34396, R0                          ; 1855
                          0251  31 001DA 31$:         BRW     71$                                 ;
                            7E  D4 001DD 32$:         CLRL    -(SP)                               ; 1857
                         0000G  30 001DF              BSBW    RM$KEY_DESC                         ;
                    5E          04  C0 001E2          ADDL2   #4, SP                              ;
                    F2          50  E9 001E5          BLBC    STATUS, 31$                         ;
                1E  04  A9      0D  E5 001E8 33$:     BBCC    #13, 4(IRAB), 36$                   ; 1871
                    0B      1C  AE  E9 001ED          BLBC    REPOS_STATUS, 34$                   ; 1879
                    52    0080  C9  3C 001F1          MOVZWL  128(IRAB), R2                       ; 1881
                    51      78  A9  D0 001F6          MOVL    120(IRAB), R1                       ;
                                0C  11 001FA          BRB     35$                                 ;
                    51    00B0  C9  D0 001FC 34$:     MOVL    176(IRAB), R1                       ; 1883
                                08  13 00201          BEQL    36$                                 ;
                    52    00BC  C9  3C 00203          MOVZWL  188(IRAB), R2                       ; 1885
                         0000G  30 00208 35$:         BSBW    RM$UNLOCK                           ;
                0C2 C9      21  A7  90 0020B 36$:     MOVB    33(IDX_DFN), 194(IRAB)              ; 1888
                    50      1E  A8  90 00211          MOVB    30(RAB), RAC                        ; 1904
                0A  05  A9      04  E1 00215          BBC     #4, 5(IRAB), 37$                    ; 1906
                    50    00C3  C9  90 0021A          MOVB    195(IRAB), RAC                      ; 1918
                                03  13 0021F          BEQL    37$                                 ; 1920
                    50          02  90 00221          MOVB    #2, RAC                             ; 1922
                02  00          50  8F 00224 37$:     CASEB   RAC, #0, #2                         ; 1926
              0010      000B  0006     00228 38$:     .WORD   39$-38$,-                           ;
                                                              40$-38$,-                           ;
                                                              41$-38$                             ;
                         0000G  30 0022E 39$:         BSBW    RM$POS_SEQ                          ; 1931
                            08  11 00231              BRB     42$                                 ;
                         0000G  30 00233 40$:         BSBW    RM$POS_KEY                          ; 1935
                            03  11 00236              BRB     42$                                 ;
                         0000G  30 00238 41$:         BSBW    RM$POS_RFA                          ; 1939
                  14  AE        50  D0 0023B 42$:     MOVL    R0, STATUS                          ;
                      03    14  AE  E8 0023F          BLBS    STATUS, 43$                         ; 1926
                          00AE  31 00243              BRW     48$                                 ;
                         0000G  30 00246 43$:         BSBW    RM$RECORD_ID                        ; 1949
                0080  C9        50  B0 00249          MOVW    R0, 128(IRAB)                       ;
                      5C        03  D0 0024E          MOVL    #3, AP                              ; 1951
                         0000G  30 00251              BSBW    RM$RECORD_VBN                       ; 1958
                  78  A9        50  D0 00254          MOVL    R0, 120(IRAB)                       ;
                            00C2  C9  95 00258        TSTB    194(IRAB)                           ; 1967
                                27  12 0025C          BNEQ    45$                                 ;
                      03    00B7  CA  91 0025E        CMPB    183(IFAB), #3                       ; 1979
                                0F  1F 00263          BLSSU   44$                                 ;
                            1E  A8  95 00265          TSTB    30(RAB)                             ; 1981
                                0A  12 00268          BNEQ    44$                                 ;
            05  1C  A7          06  E1 0026A          BBC     #6, 28(IDX_DFN), 44$                ; 1983
            13  43  A9          01  E1 0026F          BBC     #1, 67(IRAB), 46$                   ; 1985
                                5C  D4 00274 44$:     CLRL    AP                                  ; 1992
```

```
                                50    00B4   CA  3C 00276          MOVZWL  180(IFAB), R0                          ; 1993
                                      60 B940  9F 0027B          PUSHAB  @96(IRAB)[R0]                          ;
                                         0000G 30 0027F          BSBW    RM$RECORD_KEY                          ;
                                5E       04    C0 00282          ADDL2   #4, SP                                 ;
                                         5C    11 00285 45$:     BRB     47$                                    ; 1979
                                         51    D4 00287 46$:     CLRL    R1                                     ; 2012
                                         0000G 30 00289          BSBW    RM$REC_OVHD                            ;
                6E              50       56    C1 0028C          ADDL3   REC_ADDR, R0, KEY                      ;
                50              6E       01    C1 00290          ADDL3   #1, KEY, R0                            ; 2014
                        10      AE       60    9A 00294          MOVZBL  (R0), 16(SP)                           ;
                        04      AE    00 BE    9A 00298          MOVZBL  @KEY, 4(SP)                            ; 2015
                        0C      AE    20 A7    9A 0029D          MOVZBL  32(IDX_DFN), 12(SP)                    ; 2017
                                50    00B4   CA  3C 002A2          MOVZWL  180(IFAB), R0                          ;
                        08      AE    60 B940  9E 002A7          MOVAB   @96(IRAB)[R0], 8(SP)                   ;
                7E      04      AE       01    C1 002AD          ADDL3   #1, 4(SP), -(SP)                       ;
                                6E    04 AE    C0 002B2          ADDL2   KEY, (SP)                              ;
0C  AE          9E      60      B9    14 AE    2C 002B6          MOVC5   20(SP), @96(IRAB), @(SP)+, 12(SP), @8(SP)  ;
                                      08 BE       002BE                                                        ;
                                         21    18 002C0          BGEQ    47$                                    ;
                        08      AE    10 AE    C0 002C2          ADDL2   16(SP), 8(SP)                          ;
                        0C      AE    10 AE    C2 002C7          SUBL2   16(SP), 12(SP)                         ;
                7E      04      AE       01    C1 002CC          ADDL3   #1, 4(SP), -(SP)                       ;
                                6E    04 AE    C0 002D1          ADDL2   KEY, (SP)                              ;
                7E      04      AE       02    C1 002D5          ADDL3   #2, KEY, -(SP)                         ;
0C  AE          9E              9E    0C AE    2C 002DA          MOVC5   12(SP), @(SP)+, @(SP)+, 12(SP), @8(SP)  ;
                                      08 BE       002E1                                                        ;
                13      06      AA       03    E1 002E3 47$:     BBC     #3, 6(IFAB), 49$                       ; 2023
                06      00A2    CA       03    E1 002E8          BBC     #3, 162(IFAB), 48$                     ; 2025
                07      00A2    CA       02    E0 002EE          BBS     #2, 162(IFAB), 49$                     ; 2027
                                50    14 AE    D0 002F4 48$:     MOVL    STATUS, R0                             ; 2029
                                         0133  31 002F8          BRW     71$                                    ;
                                         5C    D4 002FB 49$:     CLRL    AP                                     ; 2045
06              06      A8               04    E1 002FD          BBC     #4, 6(RAB), 50$                        ; 2056
09              00A2    CA               02    E1 00302          BBC     #2, 162(IFAB), 51$                     ; 2058
                        13            06 AA    E8 00308 50$:     BLBS    6(IFAB), 52$                           ; 2061
0E              04      A8               02    E0 0030C          BBS     #2, 4(RAB), 52$                        ; 2063
                        52         0080  C9    3C 00311 51$:     MOVZWL  128(IRAB), R2                          ; 2065
                        51         78 A9    D0 00316          MOVL    120(IRAB), R1                          ;
                                         0000G 30 0031A          BSBW    RM$QUERY_LCK                           ;
                                         0C    11 0031D          BRB     53$                                    ;
                        52         0080  C9    [illegible]1F 52$:  MOVZWL  128(IRAB), R2                          ; 2067
                        51         78 A9    D[illegible]24          MOVL    120(IRAB), R1                          ;
                                         0000G 30 00328          BSBW    RM$LOCK                                ;
                        52               50    D0 0032B 53$:     MOVL    R0, STATUS                             ;
        00008061        8F               52    D1 0032E          CMPL    STATUS, #32865                         ; 2076
                                         67    12 00335          BNEQ    60$                                    ;
                        0A         06 AA    E9 00337          BLBC    6(IFAB), 54$                           ; 2088
04              00A0    CA               01    E1 0033B          BBC     #1, 160(IFAB), 54$                     ; 2090
                40      A9               01    90 00341          MOVB    #1, 64(IRAB)                           ; 2092
                                         7E    D4 00345 54$:     CLRL    -(SP)                                  ; 2094
                        7E         0080  C9    3C 00347          MOVZWL  128(IRAB), -(SP)                       ; 2095
                                      78 A9    DD 0034C          PUSHL   120(IRAB)                              ; 2094
                                         0000G 30 0034F          BSBW    RM$FIND_BY_RRV                         ;
                        5E               0C    C0 00352          ADDL2   #12, SP                                ;
                        03               50    E8 00355          BLBS    TEMP_STATUS, 55$                       ;
                        52               50    D0 00358          MOVL    TEMP_STATUS, STATUS                    ; 2098
                        3D               52    E9 0035B 55$:     BLBC    STATUS, 59$                            ; 2105
```

```
3C          66          05  E0 0035E          BBS     #5, (REC_ADDR), 60$                  ; 2107
            36          52  E9 00362          BLBC    STATUS, 59$                          ; 2121
                    21  A7  95 00365          TSTB    33(IDX_DFN)                          ; 2118
                        20  12 00368          BNEQ    58$                                  ;
            50      20  A9  D0 0036A          MOVL    32(IRAB), R0                         ; 2121
        70  A9      1C  A0  D0 0036E          MOVL    28(R0), 112(IRAB)                    ;
            03    00B7  CA  91 00373          CMPB    183(IFAB), #3                        ; 2122
                        06  1E 00378          BGEQU   56$                                  ;
            50      01  A6  9A 0037A          MOVZBL  1(REC_ADDR), R0                      ;
                        04  11 0037E          BRB     57$                                  ;
            50      01  A6  3C 00380 56$:     MOVZWL  1(REC_ADDR), R0                      ;
        74  A9          50  B0 00384 57$:     MOVW    R0, 116(IRAB)                        ;
                        14  11 00388          BRB     60$                                  ; 2118
            54      20  A9  D0 0038A 58$:     MOVL    32(IRAB), BDB                        ; 2144
                    20  A9  D4 0038E          CLRL    32(IRAB)                             ;
                        7E  D4 00391          CLRL    -(SP)                                ;
                     0000G  30 00393          BSBW    RM$RLSBKT                            ;
            5E          04  C0 00396          ADDL2   #4, SP                               ;
                        03  11 00399          BRB     60$                                  ; 2118
                    20  A9  D4 0039B 59$:     CLRL    32(IRAB)                             ; 2147
            44          52  E9 0039E 60$:     BLBC    STATUS, 65$                          ; 2156
                    20  A9  D5 003A1          TSTL    32(IRAB)                             ; 2158
                        3F  13 003A4          BEQL    65$                                  ;
04          66          05  E0 003A6          BBS     #5, (REC_ADDR), 61$                  ; 2160
37          66          06  E1 003AA          BBC     #6, (REC_ADDR), 65$                  ; 2162
            53      01  A6  B0 003AE 61$:     MOVW    1(REC_ADDR), RECORD_ID               ; 2169
            06      06  AA  E9 003B2          BLBC    6(IFAB), 62$                         ; 2175
            00000000G   EF  16 003B6          JSB     RM$RU_RECLAIM                        ; 2177
        01  A6          53  B1 003BC 62$:     CMPW    RECORD_ID, 1(REC_ADDR)               ; 2189
                        04  12 003C0          BNEQ    63$                                  ;
1F          66          05  E1 003C2          BBC     #5, (REC_ADDR), 65$                  ; 2191
            54      20  A9  D0 003C6 63$:     MOVL    32(IRAB), BDB                        ; 2198
                    20  A9  D4 003CA          CLRL    32(IRAB)                             ;
                        7E  D4 003CD          CLRL    -(SP)                                ;
                     0000G  30 003CF          BSBW    RM$RLSBKT                            ;
            5E          04  C0 003D2          ADDL2   #4, SP                               ;
      8061  8F          52  B1 003D5          CMPW    STATUS, #32865                       ; 2200
                        07  12 003DA          BNEQ    64$                                  ;
            52    8262  8F  3C 003DC          MOVZWL  #33378, STATUS                       ; 2202
                        02  11 003E1          BRB     65$                                  ;
                        52  D4 003E3 64$:     CLRL    STATUS                               ; 2204
06        06  A8          04  E1 003E5 65$:     BBC     #4, 6(RAB), 66$                  ; 2210
3B      00A2  CA          02  E1 003EA          BBC     #2, 162(IFAB), 70$               ; 2212
            05      06  AA  E8 003F0 66$:     BLBS    6(IFAB), 67$                         ; 2214
32        04  A8          02  E1 003F4          BBC     #2, 4(RAB), 70$                  ; 2216
21        04  A9          0D  E4 003F9 67$:     BBSC    #13, 4(IRAB), 68$                ; 2231
            01          52  B1 003FE          CMPW    STATUS, #1                           ; 2233
                        1C  13 00401          BEQL    68$                                  ;
      8061  8F          52  B1 00403          CMPW    STATUS, #32865                       ; 2235
                        15  13 00408          BEQL    68$                                  ;
      8071  8F          52  B1 0040A          CMPW    STATUS, #32881                       ; 2237
                        0E  13 0040F          BEQL    68$                                  ;
                    20  A9  D5 00411          TSTL    32(IRAB)                             ; 2257
                        15  12 00414          BNEQ    70$                                  ;
      82AA  8F          52  B1 00416          CMPW    STATUS, #33450                       ; 2259
                        0A  12 0041B          BNEQ    69$                                  ;
                        0C  11 0041D          BRB     70$                                  ; 2263
```

```
                        5C          01  D0 0041F 68$:   MOVL    #1, AP                          ; 2266
                 04     06  A8      02  E0 00422        BBS     #2, 6(RAB), 70$                 ; 2268
                        05  A9      20  88 00427 69$:   BISB2   #32, 5(IRAB)                    ; 2270
                        50          52  D0 0042B 70$:   MOVL    STATUS, R0                      ; 2272
                        5E          18  C0 0042E 71$:   ADDL2   #24, SP                         ; 2275
                                        05 00431        RSB                                     ;

; Routine Size:  1074 bytes,    Routine Base:  RM$RMS3 + 032D

; 2215          2276  1 END                                   ! of module
; 2216          2277  1
; 2217          2278  0 ELUDOM



;                                PSECT SUMMARY
;
;       Name                    Bytes                       Attributes
;
;  RM$RMS3                        1887  NOVEC,NOWRT,  RD ,  EXE,NOSHR,  GBL,  REL,  CON,  PIC,ALIGN(2)


;                             Library Statistics
;
;                                      -------- Symbols --------      Pages       Processing
;       File                           Total    Loaded   Percent      Mapped      Time
;
;  _$255$DUA28:[RMS.OBJ]RMS.L32;1       3109       140         4        154         00:00.4



;                                     COMMAND QUALIFIERS

;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:RM3GET/OBJ=OBJ$:RM3GET MSRC$:RM3GET/UPDATE=(ENH$:RM3GET)

; Size:          1887 code + 0 data bytes
; Run Time:         00:48.3
; Elapsed Time:     01:24.0
; Lines/CPU Min:     2829
; Lexemes/CPU-Min: 17134
; Memory Used:  399 pages
; Compilation Complete
```

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

RM3MAKIDX
LIS

RM3FNDRRV
LIS

RM3FNDRFA
LIS

RM3GET
LIS

RM3IUDR
LIS

RM3JOURNL
LIS

RM3KEYDSC
LIS

RM3MISC
LIS

RM3MISPUT
LIS