```
RRRRRRRRRRR     MMM       MMM  SSSSSSSSSSSS
RRRRRRRRRRR     MMM       MMM  SSSSSSSSSSSS
RRRRRRRRRRR     MMM       MMM  SSSSSSSSSSSS
RRR        RRR  MMMMMM MMMMMM SSS
RRR        RRR  MMMMMM MMMMMM SSS
RRR        RRR  MMMMMM MMMMMM SSS
RRR        RRR  MMM   MMM   MMM SSS
RRR        RRR  MMM   MMM   MMM SSS
RRR        RRR  MMM   MMM   MMM SSS
RRRRRRRRRRR     MMM       MMM    SSSSSSSSS
RRRRRRRRRRR     MMM       MMM    SSSSSSSSS
RRRRRRRRRRR     MMM       MMM    SSSSSSSSS
RRR   RRR       MMM       MMM             SSS
RRR   RRR       MMM       MMM             SSS
RRR   RRR       MMM       MMM             SSS
RRR      RRR    MMM       MMM             SSS
RRR      RRR    MMM       MMM             SSS
RRR      RRR    MMM       MMM             SSS
RRR        RRR  MMM       MMM SSSSSSSSSSSS
RRR        RRR  MMM       MMM SSSSSSSSSSSS
RRR        RRR  MMM       MMM SSSSSSSSSSSS
```

FILEID**RM3DELETE

```
RRRRRRRR    MM      MM  333333   DDDDDDDD    EEEEEEEEEE  LL          EEEEEEEEEE  TTTTTTTTTT  EEEEEEEEEE
RRRRRRRR    MM      MM  333333   DDDDDDDD    EEEEEEEEEE  LL          EEEEEEEEEE  TTTTTTTTTT  EEEEEEEEEE
RR      RR  MMMM  MMMM 33      33 DD      DD  EE          LL          EE              TT      EE
RR      RR  MMMM  MMMM 33      33 DD      DD  EE          LL          EE              TT      EE
RR      RR  MM  MM  MM         33 DD      DD  EE          LL          EE              TT      EE
RR      RR  MM  MM  MM         33 DD      DD  EE          LL          EE              TT      EE
RRRRRRRR    MM      MM       33   DD      DD  EEEEEEEE    LL          EEEEEEEE        TT      EEEEEEEE
RRRRRRRR    MM      MM       33   DD      DD  EEEEEEEE    LL          EEEEEEEE        TT      EEEEEEEE
RR  RR      MM      MM         33 DD      DD  EE          LL          EE              TT      EE
RR  RR      MM      MM         33 DD      DD  EE          LL          EE              TT      EE
RR    RR    MM      MM 33      33 DD      DD  EE          LL          EE              TT      EE            ....
RR    RR    MM      MM 33      33 DD      DD  EE          LL          EE              TT      EE            ....
RR      RR  MM      MM  333333   DDDDDDDD    EEEEEEEEEE  LLLLLLLLLL  EEEEEEEEEE      TT      EEEEEEEEEE    ....
RR      RR  MM      MM  333333   DDDDDDDD    EEEEEEEEEE  LLLLLLLLLL  EEEEEEEEEE      TT      EEEEEEEEEE    ....

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
```

```
MODULE RM3DELETE (LANGUAGE (BLISS32) ,
                IDENT = 'V04-000'
                ) =
BEGIN
!
!****************************************************************************
!*                                                                          *
!*  COPYRIGHT (c) 1978, 1980, 1982, 1984 BY                                 *
!*  DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.                  *
!*  ALL RIGHTS RESERVED.                                                    *
!*                                                                          *
!*  THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED   *
!*  ONLY IN  ACCORDANCE WITH  THE  TERMS  OF  SUCH  LICENSE  AND WITH THE   *
!*  INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR  ANY  OTHER   *
!*  COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY   *
!*  OTHER PERSON.  NO TITLE TO AND OWNERSHIP OF  THE  SOFTWARE IS  HEREBY   *
!*  TRANSFERRED.                                                            *
!*                                                                          *
!*  THE INFORMATION IN THIS SOFTWARE IS  SUBJECT TO CHANGE WITHOUT NOTICE   *
!*  AND  SHOULD  NOT  BE  CONSTRUED AS  A COMMITMENT BY DIGITAL EQUIPMENT   *
!*  CORPORATION.                                                            *
!*                                                                          *
!*  DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE  OR  RELIABILITY OF ITS   *
!*  SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.                 *
!*                                                                          *
!*                                                                          *
!****************************************************************************

!++
!
! FACILITY:     RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
!
! ABSTRACT:
!       This module handles the deletion of index sequential records.
!
!
!
! ENVIRONMENT:
!
!               VAX/VMS OPERATING SYSTEM
!
!--

!
! AUTHOR:       Todd M. Katz        CREATION DATE:      14-Jul-1982
!
!
! MODIFIED BY:
!
!       V03-025 JWT0181             Jim Teague              15-May-1984
!               RM$SQUISH moves too many bytes when squishing the
!               the data portion out of deleted records.
!
!       V03-024 DAS0001             David Solomon           25-Mar-1984
!               Fix broken branch to RM$RU_JOURNAL3.
!
!       V03-023 MCN0003             Maria del C. Nasr       04-Apr-1983
```

```
1 !                    Change linkage of RM$NULLKEY to RL$JSB.
1 !
1 !     V03-022 TMK0013            Todd M. Katz             26-Mar-1983
1 !             Change the linkage for RM$RU_JOURNAL3 from RL$RABREG_467 to
1 !             RL$RABREG_67.
1 !
1 !     V03-021 MCN0002            Maria del C. Nasr        24-Mar-1983
1 !             More linkages reorganization.
1 !
1 !     V03-020 RAS0135            Ron Schaefer             17-mar-1983
1 !             Fix spelling of RJR$_DELET -> RJR$_DELETE.
1 !
1 !     V03-019 TMK0012            Todd M. Katz             16-Mar-1983
1 !             Change the linkage for RM$RU_JOURNAL3 from RL$RABREG_67 to
1 !             RL$RABREG_467.
1 !
1 !     V03-018 TMK0011            Todd M. Katz             16-Mar-1983
1 !             Change the symbol RMSR$_DELET to RJR$_DELET.
1 !
1 !     V03-017 MCN0001            Maria del C. Nasr        24-Feb-1983
1 !             Reorganize linkages
1 !
1 !     V03-016 TMK0010            Todd M. Katz             08-Jan-1983
1 !             Add support for Recovery Unit Journalling and RU ROLLBACK
1 !             Recovery of ISAM files. This support includes:
1 !
1 !             1. The restructuring of RM$DELETE3B so that the primary data
1 !                record is unpacked and available for RU journalling before
1 !                any part of the file is permanently modified.
1 !
1 !             2. The RU Journalling of all $DELETEs which occur on RU
1 !                Journalled files within Recovery Units.
1 !
1 !             3. Modifications to RM$DELETE_RRV, RM$SQUISH_SIDR, and
1 !                RM$DELETE_UDR so that no space is reclaimed when records of
1 !                RU journalled files are $DELETEd within Recovery Units. The
1 !                RRV, primary data record, or SIDR array element is just
1 !                marked RU_DELETE instead.
1 !
1 !             4. Modifications to RM$DELETE_RRV, RM$SQUISH_SIDR, and
1 !                RM$DELETE_UDR so that RRVs, primary data records and SIDR
1 !                array elements maybe un-deleted during ROLLBACK of
1 !                prematurely terminated or aborted Recovery Units.
1 !
1 !             5. The addition of a second parameter (SCAN) to RM$SQUISH_SIDR.
1 !                If this parameter is 1 on entry, RMS will scan the entire
1 !                SIDR array looking for non-deleted elements even if no
1 !                duplicates are allowed in the key of reference. If SCAN is 0
1 !                RMS will immediately delete the entire SIDR as was the case
1 !                previously.
1 !
1 !     V03-015 TMK0009            Todd M. Katz             05-Jan-1983
1 !             The routine RM$DELETE_SIDR no longer calls the routine
1 !             RM$FND_SDR_ARRY to position to the SIDR element it is to
1 !             delete. It now performs its own positioning.
1 !
1 !     V03-014 TMK0008            Todd M. Katz             07-Dec-1982
```

```
1 !                     Change the order in which the various parts of a record are
1 !                     deleted during a $DELETE. First, eliminate the RRV. Next
1 !                     eliminate the user data record. Finally, the alternate keys
1 !                     which are represented in the primary data record are removed.
1 !                     Previously, the SIDRs were eliminated before the primary data
1 !                     record, and during this time a lock was kept on the the primary
1 !                     data bucket. This meant that a bucket lock was being held for
1 !                     quite a long time, and that the routine that positioned to a
1 !                     primary data record by means of an alternate index had to be
1 !                     enhanced with a very complex and very large SIDR re-positioning
1 !                     routine, so that the 1.5 SIDR deadlock case would not exist in
1 !                     version 4. Changing the order of events that take place during
1 !                     a $DELETE allowed a change in the bucket lock strategy which
1 !                     had the dual benefits of eliminating the 1.5 SIDR deadlock
1 !                     case without the expensive SIDR re-positioning code, and
1 !                     reducing the amount of time a lock on the primary data bucket
1 !                     is kept to a minimum - which is an overall ISAM design goal.
1 !
1 !                     This change is not without its cost. The reason why the old
1 !                     strategy was orginally implemented, was so that the primary
1 !                     data record would be available for the extraction of the
1 !                     alternate keys so that the corresponding SIDRs could be
1 !                     eliminated. Changing the bucket locking strategy such that
1 !                     the primary data record is deleted and the bucket is released
1 !                     before the SIDRs are deleted means that the primary data record
1 !                     must be saved in an auxillary record buffer before it is deleted
1 !                     so that it will be available for alternate key extraction.
1 !                     However, this change is not as expensive as it might seem
1 !                     because if the file's prologue version is 3, the primary data
1 !                     record would have to be unpacked into this same record buffer
1 !                     before the keys could be extracted anyway. Thus, it was a
1 !                     simple matter of unpacking either sooner or later. Any
1 !                     additional cost incurred by this new strategy is born solely by
1 !                     prologue 1 and 2 files which previously could extract the
1 !                     alternate keys without moving the primary data record, and now
1 !                     must perform an additional MOVC3. However, the benefits derived
1 !                     from this new strategy more than outweigh the cost of this
1 !                     additional MOVC3 required in the case of a prologue version
1 !                     which will hopefully fade out of use.
1 !
1 !             V03-013 TMK0007           Todd M. Katz            06-Dec-1982
1 !                     The routine RM$SQUISH_SIDR was recovering the space occupied
1 !                     by a SIDR whenever duplicates were allowed and all the elements
1 !                     in the SIDR were deleted even if the SIDR occupied the
1 !                     physically last position in the SIDR bucket. This had the
1 !                     possibility of creating totally empty SIDR buckets, and the
1 !                     encountering of a totally empty SIDR bucket during a
1 !                     positioning for insertion when duplicates are allowed can not
1 !                     always be correctly handled. Thus, a bug existed in the
1 !                     $DELETE code which had capability of corrupting SIDR indicies.
1 !
1 !                     To fix this code I have decided that the space occupied by the
1 !                     physically last SIDR in the bucket can never be recovered even
1 !                     if all the elements in the array are deleted when duplicates
1 !                     alternate keys are allowed. At best, if the file is a prologue
1 !                     3 file, and the element is not the first element in the SIDR
1 !                     array, the space occupied by the RRV pointer can be recovered.
```

```
This fix which I have implemented by re-writing the routine
RM$SQUISH_SIDR (both to implement the fix and to optimize the
existing code) guarentees both that empty SIDR buckets can
never be created when duplicate SIDRs are allowed, and that NRP
positioning context is maintained.

V03-012 TMK0006         Todd M. Katz            14-Nov-1982
        The routine RM$DELETE_UDR no longer has to return a value.
        Previously, it was returning a value because the routine that
        was responsible for reclaiming space occupied by records that
        were just marked deleted needed to know whether or no an
        RRV had been created in the place of the reclaimed record.
        This is no longer the case, as that routine has been modified
        to no longer require this piece of information.

V03-011 TMK0005         Todd M. Katz            12-Nov-1982
        The routine RM$FND_SDR_ARRY requires as implicit input the key
        size of the SIDR it is to position to in IRB$B_KEYSZ. The
        routine RM$DELETE_SIDR was not setting up the IRAB cell with
        the key size before calling this routine. Therefore, the
        possibility existed that RM$FND_SDR_ARRY would position to
        the wrong SIDR array, which would then be deleted. This in fact
        has been seen, during the course of an $UPDATE when the old
        SIDRs that have been changed are removed, and this fix corrects
        this problem.

V03-010 TMK0004         Todd M. Katz            11-Nov-1982
        When SIDRs must be deleted and the file is a prologue 3 file,
        the record must be unpacked so that the alternate keys can be
        extracted. If RMS positioned by the primary key of reference
        then it will already have a fully expanded copy of the primary
        key in keybuffer 1, and it can use this in the unpacking of the
        record instead of scanning the bucket to re-expand the primary
        key when primary key compression is enabled. There is one
        case when it can not use the primary key in keybuffer 1 like
        this, and that is when the record being deleted is not the same
        as the current primary data record. This happens when RMS
        randomly $FINDs a record since this operation does not update
        the NRP conext. I was not checking for this case and this fix
        remedies this.

V03-009 TMK0003         Todd M. Katz            06-Oct-1982
        When I completely re-wrote this routine (TMK0001), I broke
        the deletion of prologue 3 fixed length records, in certain
        cases, because I had assummed that all prologue 3 records
        included as part of their record overhead a record size field
        that needs to be updated when the portion of the prologue 3
        primary data record occupied by the data is reclaimed. I thought
        I had fixed this in TMK0002 (although I forgot to mention it in
        the audit trial), but actually all I did was fix one $DELETE
        case and break others that occur more frequently. What I did
        was to make the assunmption that all fixed length prologue 3
        records do not include a record size field. This too is
        incorrect. Actually, if a prologue 3 record with fixed length
        records has either key or data compression (or both) enabled,
        then there is a record size field present as part of the
        record overhead. If both compression types are disabled and the
```

```
1 !                record is fixed size then there is no need for a record size
1 !                field and one is not present. I was not checking any compression
1 !                bits, but rather, just for a fixed length record format, before
1 !                deciding whether or there was a record size field to update and
1 !                this is what caused the problem in TMK0002.
1 !
1 !        V03-008 TMK0002          Todd M. Katz              04-Sep-1982
1 !                Add support for prologue 3 SIDRs. This involves changes
1 !                only to the routine RM$SQUISH_SIDR.
1 !
1 !                The field IRB$B_SRCHFLAGS is now a word. Change all references
1 !                to it.
1 !
1 !        V03-007 KBT0162          Keith B. Thompson         21-Aug-1982
1 !                Reorganize psects
1 !
1 !        V03-006 TMK0001          Todd M. Katz              02-Jul-1982
1 !
1 !                New version of $DELETE. This module now incorporates all
1 !                the routines which were formerly in RM3DELSDR.
1 !
1 !*****
1
1 LIBRARY 'RMSLIB:RMS';
1
1 REQUIRE 'RMSSRC:RMSIDXDEF';
1
1 ! Define default PSECTS for code.
1 !
1 PSECT
1     CODE = RM$RMS3(PSECT_ATTR),
1     PLIT = RM$RMS3(PSECT_ATTR);
1
1 ! Linkages.
1 !
1 LINKAGE
1     L_ERROR_LINK1,
1     L_JSB,
1     L_JSB01,
1     L_LINK_7_10_11,
1     L_PRESERVE1,
1     L_RABREG,
1     L_RABREG_4567,
1     L_RABREG_567,
1     L_RABREG_67,
1     L_RABREG_7,
1     L_REC_OVHD,
1     L_SIDR_FIRST,
1
1     ! Local Linkage
1     !
1     RL$DEL_ALL_SIDR = JSB ()
1                 : GLOBAL (R_REC_ADDR,R_IDX_DFN,COMMON_RABREG),
1     RL$SQUISH_DATA = JSB ()
1                 : GLOBAL(R_REC_ADDR,R_BKT_ADDR,R_IDX_DFN,R_IFAB);
1
1 ! External Routines
```

```
1 !
1 EXTERNAL ROUTINE
1       RM$CLEAN_BDB            : RL$ERROR_LINK1,
1       RM$CSEARCH_TREE         : RL$RABREG_67,
1       RM$EXPAND_KEYD          : RL$JSB01,
1       RM$EXT_ARRY_RFA         : RL$RABREG_67,
1       RM$FIND_BY_ID           : RL$RABREG_567,
1       RM$GET_NEXT_KEY         : RL$LINK_7_10_11,
1       RM$GETNXT_ARRAY         : RL$RABREG_67,
1       RM$KEY_DESC             : RL$RABREG_7,
1       RM$MOVE                 : RL$PRESERVE1,
1       RM$NULLKEY              : RL$JSB,
1       RM$REC_OVHD             : RL$REC_OVHD,
1       RM$RECORD_ID            : RL$RABREG_67,
1       RM$RECORD_KEY           : RL$PRESERVE1,
1       RM$RECORD_VBN           : RL$PRESERVE1,
1       RM$RLSBKT               : RL$PRESERVE1,
1       RM$RU_JOURNAL3          : RL$RABREG_67 ADDRESSING_MODE( LONG_RELATIVE ),
1       RM$SIDR_END             : RL$RABREG_67,
1       RM$SIDR_FIRST           : RL$SIDR_FIRST,
1       RM$UNPACK_REC           : RL$JSB01,
1       RM$UPDDELCOM            : RL$RABREG_67;
1
1 ! Forward Routines
1 !
1 FORWARD ROUTINE
1       RM$DELETE_RRV           : RL$RABREG_4567,
1       RM$DELETE_SIDR          : RL$RABREG_7,
1       RM$DELETE_UDR           : RL$RABREG_4567  NOVALUE,
1       RM$SQUISH_DATA          : RL$SQUISH_DATA NOVALUE,
1       RM$SQUISH_SIDR          : RL$RABREG_567;
```

```
%SBTTL  'RM$DEL_ALL_SIDR'
ROUTINE RM$DEL_ALL_SIDR (RECORD_SIZE) : RL$DEL_ALL_SIDR NOVALUE =

!++
!
! FUNCTIONAL DESCRIPTION:
!
!       The purpose of this routine is to delete every SIDR array element
!       pointing to the the current primary data record. Towards this goal
!       every secondary key represented in the current primary data record
!       is in turn extracted from the current primary data record which has
!       been saved (in an unpacked form if prologue 3) in a record buffer, used
!       to position to the SIDR array element pointing to the current primary
!       data record in the appropriate index, and that array element is
!       deleted. If the current primary data record does not possess one or
!       more secondary keys either because the record is not of sufficient size
!       or the key is null, or if a fast delete is requested and duplicates of
!       one or more secondary keys are allowed, then the deletion of those
!       secondary keys are bypassed.
!
! CALLING SEQUENCE:
!
!       RM$DEL_ALL_SIDR()
!
! INPUT PARAMETERS:
!
!       RECORD_SIZE             - size of the user data record in IRB$L_RECBUF
!
! IMPLICIT INPUTS:
!
!       IDX_DFN                         - index descriptor for the primary key
!
!       IFAB                            - address of the IFAB
!           IFB$W_KBUFSZ                - size of each of the keybuffers
!           IFB$B_PLG_VER               - prologue version of the file
!
!       IRAB                            - address of the IRAB
!           IRB$L_KEYBUF                - address of the contigious keybuffers
!           IRB$L_RECBUF                - address of record unpacking buffer
!
!       RAB                             - address of the RAB
!           RAB$V_FDL                   - if set, fast-delete requested
!
! OUTPUT PARAMETERS:
!       NONE
!
! IMPLICIT OUTPUTS:
!       NONE
!
! ROUTINE VALUE:
!       NONE
!
! SIDE EFFECTS:
!
!       AP and REC_ADDR are trashed.
!       Keybuffer 2 contains the key of the last SIDR deleted.
!
```

```
;  1 !--
;  1
;  2        BEGIN
;  2
;  2        BUILTIN
;  2            AP;
;  2
;  2        EXTERNAL REGISTER
;  2            COMMON_RAB_STR,
;  2            R_IDX_DFN_STR,
;  2            R_REC_ADDR_STR;
;  2
;  2        LABEL
;  2            BLOCK;
;  2
;  2        ! Delete all of the secondary keys present in the current user data record.
;  2        !
;  2        WHILE RM$GET_NEXT_KEY()
;  2        DO
;  2
;  2            ! Each secondary key in the file will in turn become the "current"
;  2            ! secondary key for the purpose of deleting its representative in the
;  2            ! current primary data record from the appropriate index.
;  2            !
;  3 BLOCK:     BEGIN
;  3
;  3            ! If a fast-delete is requested, terminate the deletion of the current
;  3            ! secondary key only if this secondary key allows duplicates. If this
;  3            ! secondary key does not allow duplicates, then a fast delete of it can
;  3            ! not be done, since the error caused by a later attempt to insert a
;  3            ! record with a secondary key that is a duplicate of this one would go
;  3            ! undetected.
;  3            !
;  3            IF .RAB[RAB$V_FDL]
;  3                AND
;  3               .IDX_DFN[IDX$V_DUPKEYS]
;  3            THEN
;  3                LEAVE BLOCK;
;  3
;  3            ! Check that the current primary data record is of a sufficient size to
;  3            ! include the current secondary key. If it is not, terminate the
;  3            ! deletion process for this secondary key.
;  3            !
;  3            IF .RECORD_SIZE<0, 16> LSSU .IDX_DFN[IDX$W_MINRECSZ]
;  3            THEN
;  3                LEAVE BLOCK;
;  3
;  3            ! In preparation for positioning to the SIDR array element for this
;  3            ! secondary key of the current primary data record, the secondary key
;  3            ! must be extracted into keybuffer 2.
;  3            !
;  3            REC_ADDR = .IRAB[IRB$L_RECBUF];
;  3
;  3            ! If this secondary key for the current primary data record is null,
;  3            ! there will not be a SIDR array element in this index pointing to the
;  3            ! current primary data record. Therefore, there is no need to continue
;  3            ! with the process of deleting the current secondary key's
```

```
;  432      0495  3          ! representative in the current primary data record.
;  433      0496  3          !
;  434      0497  3          AP = 0;
;  435      0498  3          IF NOT RM$NULLKEY (.REC_ADDR)
;  436      0499  3          THEN
;  437      0500  3              LEAVE BLOCK;
;  438      0501  3
;  439      0502  3          ! Extract out the current secondary key from the current primary data
;  440      0503  3          ! record, and place it in keybuffer 2.
;  441      0504  3          !
;  442      0505  3          AP = 3;
;  443      0506  3
;  444      0507  4          BEGIN
;  445      0508  4
;  446      0509  4          GLOBAL REGISTER
;  447      0510  4              R_BDB;
;  448      0511  4
;  449      0512  4          RM$RECORD_KEY (KEYBUF_ADDR(2));
;  450      0513  3          END;
;  451      0514  3
;  452      0515  3          ! Position to and delete the SIDR array element pointing to the current
;  453      0516  3          ! primary data record for this secondary key from the file.
;  454      0517  3          !
;  455      0518  3          RM$DELETE_SIDR();
;  456      0519  2          END;
;  457      0520  2
;  458      0521  1      END;


                                                .TITLE  RM3DELETE
                                                .IDENT  \V04-000\

                                                .EXTRN  RM$CLEAN_BDB, RM$CSEARCH_TREE
                                                .EXTRN  RM$EXPAND_KEYD, RM$EXT_ARRY_RFA
                                                .EXTRN  RM$FIND_BY_ID, RM$GET_NEXT_KEY
                                                .EXTRN  RM$GETNXT_ARRAY
                                                .EXTRN  RM$KEY_DESC, RM$MOVE
                                                .EXTRN  RM$NULLKEY, RM$REC_OVHD
                                                .EXTRN  RM$RECORD_ID, RM$RECORD_KEY
                                                .EXTRN  RM$RECORD_VBN, RM$RLSBKT
                                                .EXTRN  RM$RU_JOURNAL3, RM$SIDR_END
                                                .EXTRN  RM$SIDR_FIRST, RM$UNPACK_REC
                                                .EXTRN  RM$UPDDELCOM

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

                              54  DD 00000 RM$DEL_ALL_SIDR:
                                                PUSHL   R4                                              ; 0382
                           0000G  30 00002 1$:  BSBW    RM$GET_NEXT_KEY                                 ; 0455
                  38          50  E9 00005      BLBC    R0, 3$                                          ;
   04       04    A8          06  E1 00008      BBC     #6, 4(RAB), 2$                                  ; 0471
                  F1     1C   A7  E8 0000D      BLBS    28(IDX_DFN), 1$                                 ; 0473
            22    A7     08   AE  B1 00011 2$:  CMPW    RECORD_SIZE, 34(IDX_DFN)                        ; 0481
                              EA  1F 00016      BLSSU   1$                                              ;
            56           68   A9  D0 00018      MOVL    104(IRAB), REC_ADDR                             ; 0489
                              5C  D4 0001C      CLRL    AP                                              ; 0497
                              56  DD 0001E      PUSHL   REC_ADDR                                        ; 0498
```

```
                     0000G  30 00020         BSBW    RM$NULLKEY                 ;
         5E             04  C0 00023         ADDL2   #4, SP                     ;
         D9             50  E9 00026         BLBC    R0, 1$                     ;
         5C             03  D0 00029         MOVL    #3, AP                     ; 0505
         50       00B4  CA  3C 0002C         MOVZWL  180(IFAB), R0              ; 0512
                   60 B940  9F 00031         PUSHAB  @96(IRAB)[R0]              ;
                     0000G  30 00035         BSBW    RM$RECORD_KEY              ;
         5E             04  C0 00038         ADDL2   #4, SP                     ;
                     0000V  30 0003B         BSBW    RM$DELETE_SIDR             ; 0518
                        C2  11 0003E         BRB     1$                         ; 0455
                        10  BA 00040 3$:     POPR    #^M<R4>                    ; 0521
                            05 00042         RSB                                ;

; Routine Size:  67 bytes,    Routine Base:  RM$RMS3 + 0000
```

```
1 %SBTTL  'RM$DELETE3B'
1 GLOBAL ROUTINE RM$DELETE3B : RL$RABREG =
1
1 !++
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       This routine directs the deletion of the current primary data record.
1 !       To establish a current record, a $GET or $FIND is done. Fast delete
1 !       (SIDR entries are not deleted) can only take place when duplicates are
1 !       allowed. This is because allowing SIDR entries to not be deleted when
1 !       duplicates were not allowed, would mean that the the error condition
1 !       ''inserting duplicate when not allowed'' could not be detected.
1 !
1 !       The steps involved in deleting the current record are as follows:
1 !
1 !       1. If the file defines alternate keys or is being RU Journalled, save
1 !          the primary data record in a record buffer. If the file is a
1 !          prologue 3 file then the primary data record will be saved in
1 !          unpacked format.
1 !
1 !       2. Delete the RRV. The space it occupies maybe completely reclaimed
1 !          if the file is a prologue 3 file; otherwise, just the space
1 !          occupied by the RRV pointer is recovered.
1 !
1 !       3. Delete the user data record. This may involve just marking it
1 !          deleted, eliminating just the data portion (prologue 3 only), or
1 !          eliminating the entire record depending upon the prologue version
1 !          of the file, whether duplicate primary keys are allowed, and whether
1 !          this primary data record is physically the last record in the primary
1 !          data bucket.
1 !
1 !       4. Delete all secondary keys (unless fast delete is set and duplicates
1 !          are allowed). The SIDR will be completely deleted if duplicates
1 !          are not allowed, but if duplicates are allowed the SIDR element will
1 !          just be marked deleted and the space occupied by the RRV pointer
1 !          reclaimed if the file is a prologue 3 file.
1 !
1 !       NOTE: If this operation is occurring on a RU Journalled file within a
1 !             recovery unit then the RRV, primary data record, and all SIDR
1 !             elements are marked IRC$V_RU_DELETE and no space is reclaimed.
1 !
1 ! CALLING SEQUENCE:
1 !
1 !       RM$DELETE3B()
1 !
1 ! INPUT PARAMETERS:
1 !       NONE
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       IFAB                    - address of IFAB
1 !           IFB$B_NUM_KEYS      - number of keys in the file
1 !           IFB$B_PLG_VER       - prologue version of the file
1 !           IFB$V_RUP           - if set, Recovery Unit is in progress
1 !
1 !       IRAB                    - address of IRAB
```

```
!                       IRB$B_CUR_KREF          - current positioning key of reference
!                       IRB$W_POS_ID            - ID of positioning primary data record
!                       IRB$L_POS_VBN           - VBN of positioning primary data record
!                       IRB$L_RECBUF            - address of record buffer
!                       IRB$W_UDR_ID            - ID of current primary data record
!                       IRB$L_UDR_VBN           - VBN of current primary data record
!
! OUTPUT PARAMETERS:
!       NONE
!
! IMPLICIT OUTPUTS:
!
!       IRAB
!               IRB$V_FIND_LAST         - 0, last operation was not a $FIND
!               IRB$V_PUTS_LAST         - 0, last operation was not a $PUT
!               IRB$V_UPDATE            - 0, last operation was not an $UPDATE
!
! ROUTINE VALUE:
!
!       CUR             - illegal or no current record
!       RNL             - current record not locked
!       SUC             - record successfully deleted
!       various I/O errors
!
! SIDE EFFECTS:
!
!       If record locking is unneccessary the record locks are not checked for.
!       If automatic locking is not specified, the then the deleted record is
!               not unlocked.
!       If automatic locking is required, then the current primary data record
!               is always unlocked, on success or failure.
!       If the current process is within a Recovery Unit, and the file is being
!               Recovery Unit Journalled, then the operation is RU Journalled
!               before any permanent modification to the file takes place
!--

    BEGIN

    BUILTIN
        AP;

    EXTERNAL REGISTER
        COMMON_RAB_STR;

    GLOBAL REGISTER
        COMMON_IO_STR,
        R_REC_ADDR_STR,
        R_IDX_DFN_STR;

    LOCAL
        RECORD_SIZE;

    ! Perform checks common to both $UPDATE and $DELETE such as making sure
    ! there is a current record and that it is locked, and then find the
    ! current record by means of its RFA address. This will access both the
    ! bucket containing the current record and the bucket containing the
    ! current record's RRV, if it has one. The address of the BDB for the
```

```
! current record bucket will be returned in IRB$L_CURBDB, and the address
! of the BDB for the RRV bucket will be returned in IRB$L_NXTBDB.
!
IRAB[IRB$V_UPDATE] = 0;

RETURN_ON_ERROR (RM$UPDDELCOM());

! Retrieve the index descriptor for the primary key.
!
RM$KEY_DESC (0);

! If the file contains alternate keys, then save the primary data record
! (in unpacked format if the file's prologue version is 3), in a record
! buffer so that the primary data record maybe deleted, and the record will
! still available. This is so that the alternate keys maybe extracted from
! it at a later time to be used in the deletion of the corresponding SIDRs.
!
! If the process is within a recovery unit and the file is being RU
! Journalled, then unpack the primary data record regardless of whether or
! not the file defines alternate keys.
!
IF   .IFAB[IFB$B_NUM_KEYS] GTRU 1
        OR
     .IFAB[IFB$V_RUP]
THEN
    BEGIN

    LOCAL
        REC_SIZE,
        SAVE_REC_ADDR          : REF BBLOCK;

    ! Retrieve the size of the current primary data record, and position
    ! past the record overhead to the user data record itself.
    !
    SAVE_REC_ADDR = .REC_ADDR;
    REC_ADDR = .REC_ADDR + RM$REC_OVHD(0; REC_SIZE);
    RECORD_SIZE = .REC_SIZE;

    ! If the file is a prologue 3 file, then the current primary data
    ! record must be unpacked into the record buffer
    !
    IF .IFAB[IFB$B_PLG_VER] GEQU  PLG$C_VER_3
    THEN
        BEGIN

        ! If the record is in a special format, then retrieve the true size
        ! of the record from the last two bytes in the record's reserved
        ! space.
        !
        IF .SAVE_REC_ADDR[IRC$V_RU_UPDATE]
        THEN
            RECORD_SIZE = .(.REC_ADDR + .RECORD_SIZE
                                      - IRC$C_DATSZFLD)<0,16>;

        ! As part of the process of unpacking the current primary data
        ! record, RMS must extract the primary key from its position in
        ! front of the rest of the data record, re-expand it if it is
```

```
                        ! compressed, and re-integrate it. If the current NRP positioning
                        ! key of reference is the primary key, then when RMS positioned to
                        ! the current primary data record it extracted its primary key into
                        ! keybuffer 1 where it serves as part of the local NRP context. If
                        ! this is indeed the case, then signal the data record unpacking
                        ! routine that the primary key for this data record maybe found in
                        ! keybuffer 1, and that there is no need to again extract and
                        ! re-expand the primary key as part of the unpacking process;
                        ! otherwise, signal that the entire unpacking process must be gone
                        ! through.
                        !
                        ! There is one case when RMS must signal that the entire unpacking
                        ! process must be gone through even though the primary key is the
                        ! current key of reference. This is when RMS positioned to the
                        ! record by means of a random $FIND. This type of operation does
                        ! not update the NRP context.
                        !
                        IF (.IRAB[IRB$B_CUR_KREF] EQLU 0)
                            AND
                          (.IRAB[IRB$W_POS_ID] EQLU .IRAB[IRB$W_UDR_ID])
                            AND
                          (.IRAB[IRB$L_POS_VBN] EQLU .IRAB[IRB$L_UDR_VBN])
                        THEN
                            AP = 1
                        ELSE
                            AP = 0;

                        RECORD_SIZE = RM$UNPACK_REC (.IRAB[IRB$L_RECBUF], .RECORD_SIZE);

                        ! If this file is being RU Journalled (Only Prologue 3 files are
                        ! journalled), and the current process is within a Recovery Unit,
                        ! then RU Journal the current operation and set the state bit
                        ! IRB$V_RU_DELETE ao that the deletions are done such that no space
                        ! at all is reclaimed.
                        !
                        IF .IFAB[IFB$V_RUP]
                        THEN
                            BEGIN
                            REC_ADDR = .IRAB[IRB$L_RECBUF];
                            RETURN_ON_ERROR (RM$RU_JOURNAL3 (RJR$_DELETE,
                                                             .IRAB[IRB$L_UDR_VBN],
                                                             .IRAB[IRB$W_UDR_ID],
                                                             .RECORD_SIZE),
                                              RM$CLEAN_BDB());
                            IRAB[IRB$V_RU_DELETE] = 1;
                            END;
                        END

                    ! If the file is a prologue 1 or 2 file, then just move the primary data
                    ! record into the record buffer.
                    !
                    ELSE
                        RM$MOVE (.RECORD_SIZE, .REC_ADDR, .IRAB[IRB$L_RECBUF]);

                    ! Position back to the beginning of the primary data record - to the
                    ! first byte of the current primary data record's overhead.
                    !
```

```
	    REC_ADDR = .SAVE_REC_ADDR;
	    END;

	! If the current record is not in its original bucket, process the RRV for
	! the current record. For prologue 3 files this involves deleting the RRV
	! entirely. For all other files, just the space occupied by the RRV pointer
	! to the current record is reclaimed. This means that the current record
	! can no longer be found through its secondary keys or by RFA access.
	!
	IF (BDB = .IRAB[IRB$L_NXTBDB]) NEQ 0
	THEN
	    BEGIN
	    IRAB[IRB$L_NXTBDB] = 0;
	    RETURN_ON_ERROR (RM$DELETE_RRV(), BEGIN
						IRAB[IRB$V_RU_DELETE] = 0;
						RELEASE (IRAB[IRB$L_CURBDB]);
						END);
	    END;

	! Delete the current primary data record, mark the bucket dirty and release
	! it. If the current record's key is the high key in the primary data
	! bucket, then the current primary data record is just marked deleted.
	!
	BDB = .IRAB[IRB$L_CURBDB];
	IRAB[IRB$L_CURBDB] = 0;

	RM$DELETE_UDR();

	BDB[BDB$V_DRT] = 1;
	RETURN_ON_ERROR (RM$RLSBKT(0), IRAB[IRB$V_RU_DELETE] = 0);

	! If the file contains alternate keys, delete all the SIDR entries for
	! the current record.
	!
	IF .IFAB[IFB$B_NUM_KEYS] GTRU 1
	THEN
	    RM$DEL_ALL_SIDR (.RECORD_SIZE);

	! Clear the IRB$V_RU_DELETE state bit regardless of whether this operation
	! was or wasn't RU Journalled, and then return success.
	!
	IRAB[IRB$V_RU_DELETE] = 0;
	RETURN RMSSUC();
	END;
```

```
                 00FC   8F  BB 00000 RM$DELETE3B::
                                              PUSHR   #^M<R2,R3,R4,R5,R6,R7>          ; 0523
        06   A9           08  8A 00004        BICB2   #8, 6(IRAB)                     ; 0639
                       0000G  30 00008        BSBW    RM$UPDDELCOM                    ; 0641
             03           50  E8 0000B        BLBS    STATUS, 1$                      ;
                        0101  31 0000E        BRW     13$                             ;
                          7E  D4 00011 1$:    CLRL    -(SP)                           ; 0645
                       0000G  30 00013        BSBW    RM$KEY_DESC                     ;
```

```
                          5E                04  C0 00016          ADDL2   #4, SP                          ;
                          01       00B2     CA  91 00019          CMPB    178(IFAB), #1                   ; 0657
                                            09  1A 0001E          BGTRU   2$                              ;
        03      00A2      CA                02  E0 00020          BBS     #2, 162(IFAB), 2$               ; 0659
                                          008A  31 00026          BRW     9$                              ;
                          53                56  D0 00029 2$:      MOVL    REC_ADDR, SAVE_REC_ADDR         ; 0670
                                            51  D4 0002C          CLRL    R1                              ; 0671
                                         0000G  30 0002E          BSBW    RM$REC_OVHD                     ;
                          56                50  C0 00031          ADDL2   R0, REC_ADDR                    ;
                          52                51  D0 00034          MOVL    REC_SIZE, RECORD_SIZE           ; 0672
                          03       00B7     CA  91 00037          CMPB    183(IFAB), #3                   ; 0677
                                            65  1F 0003C          BLSSU   7$                              ;
        07                63                06  E1 0003E          BBC     #6, (SAVE_REC_ADDR), 3$         ; 0685
                                   FE A246  9F 00042          PUSHAB  -2(RECORD_SIZE)[REC_ADDR]       ; 0687
                          52                9E  3C 00046          MOVZWL  @(SP)+, RECORD_SIZE             ;
                                   00C3     C9  95 00049 3$:      TSTB    195(IRAB)                       ; 0710
                                            17  12 0004D          BNEQ    4$                              ;
                00BC      C9       00BA     C9  B1 0004F          CMPW    186(IRAB), 188(IRAB)            ; 0712
                                            0E  12 00056          BNEQ    4$                              ;
                00B0      C9       00AC     C9  D1 00058          CMPL    172(IRAB), 176(IRAB)            ; 0714
                                            05  12 0005F          BNEQ    4$                              ;
                          5C                01  D0 00061          MOVL    #1, AP                          ; 0716
                                            02  11 00064          BRB     5$                              ;
                                            5C  D4 00066 4$:      CLRL    AP                              ; 0718
                          51                52  D0 00068 5$:      MOVL    RECORD_SIZE, R1                 ; 0720
                          50       68       A9  D0 0006B          MOVL    104(IRAB), R0                   ;
                                         0000G  30 0006F          BSBW    RM$UNPACK_REC                   ;
                          52                50  D0 00072          MOVL    R0, RECORD_SIZE                 ;
        35      00A2      CA                02  E1 00075          BBC     #2, 162(IFAB), 8$               ; 0728
                          56       68       A9  D0 0007B          MOVL    104(IRAB), REC_ADDR             ; 0731
                                            52  DD 0007F          PUSHL   RECORD_SIZE                     ; 0736
                          7E       00BC     C9  3C 00081          MOVZWL  188(IRAB), -(SP)                ;
                                   00B0     C9  DD 00086          PUSHL   176(IRAB)                       ;
                                            05  DD 0008A          PUSHL   #5                              ;
                                 00000000G  EF  16 0008C          JSB     RM$RU_JOURNAL3                  ;
                          5E                10  C0 00092          ADDL2   #16, SP                         ;
                          05                50  E8 00095          BLBS    STATUS, 6$                      ;
                                         0000G  30 00098          BSBW    RM$CLEAN_BDB                    ;
                                            75  11 0009B          BRB     13$                             ;
                07        A9                20  88 0009D 6$:      BISB2   #32, 7(IRAB)                    ; 0737
                                            0D  11 000A1          BRB     8$                              ; 0677
                                   68       A9  DD 000A3 7$:      PUSHL   104(IRAB)                       ; 0745
                                   0044     8F  BB 000A6          PUSHR   #^M<R2,R6>                      ;
                                         0000G  30 000AA          BSBW    RM$MOVE                         ;
                          5E                0C  C0 000AD          ADDL2   #12, SP                         ;
                          56                53  D0 000B0 8$:      MOVL    SAVE_REC_ADDR, REC_ADDR         ; 0750
                          54       3C       A9  D0 000B3 9$:      MOVL    60(IRAB), BDB                   ; 0759
                                            24  13 000B7          BEQL    10$                             ;
                                   3C       A9  D4 000B9          CLRL    60(IRAB)                        ; 0762
                                         0000V  30 000BC          BSBW    RM$DELETE_RRV                   ; 0766
                          51                50  D0 000BF          MOVL    R0, STATUS                      ;
                          18                51  E8 000C2          BLBS    STATUS, 10$                     ;
                07        A9                20  8A 000C5          BICB2   #32, 7(IRAB)                    ;
                          54       20       A9  D0 000C9          MOVL    32(IRAB), BDB                   ;
                                   20       A9  D4 000CD          CLRL    32(IRAB)                        ;
                                            7E  D4 000D0          CLRL    -(SP)                           ;
                                         0000G  30 000D2          BSBW    RM$RLSBKT                       ;
```

```
         5E              04  C0 000D5          ADDL2   #4, SP                              ;
         50              51  D0 000D8          MOVL    STATUS, R0                          ;
                         35  11 000DB          BRB     13$                                 ;
         54        20    A9  D0 000DD  10$:    MOVL    32(IRAB), BDB                       ; 0773
                   20    A9  D4 000E1          CLRL    32(IRAB)                            ; 0774
                      0000V  30 000E4          BSBW    RM$DELETE_UDR                       ; 0776
   0A    A4              02  88 000E7          BISB2   #2, 10(BDB)                         ; 0778
                         7E  D4 000EB          CLRL    -(SP)                               ; 0779
                      0000G  30 000ED          BSBW    RM$RLSBKT                           ;
         5E              04  C0 000F0          ADDL2   #4, SP                              ;
         06              50  E8 000F3          BLBS    STATUS, 11$                         ;
   07    A9              20  8A 000F6          BICB2   #32, 7(IRAB)                        ;
                         16  11 000FA          BRB     13$                                 ;
   01            00B2    CA  91 000FC  11$:    CMPB    178(IFAB), #1                       ; 0784
                         08  1B 00101          BLEQU   12$                                 ;
                         52  DD 00103          PUSHL   RECORD_SIZE                         ; 0786
                       FEB5  30 00105          BSBW    RM$DEL_ALL_SIDR                     ;
         5E              04  C0 00108          ADDL2   #4, SP                              ;
   07    A9              20  8A 0010B  12$:    BICB2   #32, 7(IRAB)                        ; 0791
         50              01  D0 0010F          MOVL    #1, R0                              ; 0792
                 00FC    8F  BA 00112  13$:    POPR    #^M<R2,R3,R4,R5,R6,R7>              ; 0793
                         05     00116          RSB                                         ;

; Routine Size:  279 bytes,    Routine Base:  RM$RMS3 + 0043
```

```
1 %SBTTL  'RM$DELETE_RRV'
1 GLOBAL ROUTINE RM$DELETE_RRV : RL$RABREG_4567 =
1
1 !++
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       Delete the RRV for the current primary data record. If the file is a
1 !       prologue 3 file the RRV is entirely deleted; otherwise, it is marked
1 !       deleted and just the space occupied by the pointer is reclaimed.
1 !
1 !       If the state bit IRB$V_RU_DELETE is set, the RRV is just marked
1 !       RU_DELETE. Likewise, if the state bit IRB$V_RU_UNDEL is set, then the
1 !       RU_DELETE bit in the RRV;s control byte is cleared.
1 !
1 ! CALLING SEQUENCE:
1 !
1 !       RM$DELETE_RRV()
1 !
1 ! INPUT PARAMETERS:
1 !       NONE
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       BDB                         - BDB of buffer with RRV bucket in it
1 !           BDB$L_ADDR              - address of buffer
1 !
1 !       IFAB                        - address of IFAB
1 !           IFB$B_PLG_VER           - prologue version of file
1 !
1 !       IRAB
1 !           IRB$V_RU_DELETE         - if set, mark RU_DELETE and do not reclaim
1 !           IRB$V_RU_UNDEL          - if set, un-delete the RRV
1 !
1 !       REC_ADDR                    - address of record whose RRV is to be deleted
1 !
1 ! OUTPUT PARAMETERS:
1 !       NONE
1 !
1 ! IMPLICIT OUTPUTS:
1 !
1 !       IDX_DFN                     - index descriptor for the primary key
1 !
1 ! ROUTINE VALUE:
1 !
1 !       Value of RLSBKT when writing out bucket with RRV deleted
1 !
1 ! SIDE EFFECTS:
1 !
1 !       AP destroyed.
1 !       IDX_DFN is set up for the primary key.
1 !       The freespace offset in the RRV bucket is updated to reflect the
1 !            amount of space reclaimed.
1 !
1 !--
1
2     BEGIN
```

```
BUILTIN
    AP;

EXTERNAL REGISTER
    R_BDB_STR,
    COMMON_RAB_STR,
    R_IDX_DFN_STR,
    R_REC_ADDR_STR;

GLOBAL REGISTER
    R_BKT_ADDR_STR;

LOCAL
    DEL_RRV_SIZE,
    LENGTH,
    RRV_SIZE,
    SAVE_REC_ADDR;

! Obtain the key descriptor for the primary key of reference.
!
RM$KEY_DESC(0);

SAVE_REC_ADDR = .REC_ADDR;

! Extract the RRV ID of the current primary data record.
!
AP = RM$RECORD_ID();

! Position to the RRV to be deleted, the RRV for the current primary data
! record. It is impossible for this positioning to fail as long as the
! bucket containing the RRV has not been released since RM$FIND_BY_RRV
! accessed it.
!
BKT_ADDR = .BDB[BDB$L_ADDR];
RM$FIND_BY_ID();

! If is is indicated that the RRV should just be marked RU_DELETE and that
! no space should be reclaimed, then do so by setting the RU_DELETE bit
! within the RRV's control byte.
!
IF .IRAB[IRB$V_RU_DELETE]
THEN
    REC_ADDR[IRC$V_RU_DELETE] = 1

! If it is indicated that the RRV should be un-deleted, then do so by
! clearing the RU_DELETE bit in the RRV's control byte.
!
ELSE
    IF .IRAB[IRB$V_RU_UNDEL]
    THEN
        REC_ADDR[IRC$V_RU_DELETE] = 0

    ! Delete the RRV reclaiming as much space as is possible.
    !
    ELSE
        BEGIN
```

```
3
3                   ! Setup a series of constants to be used in deleting the RRV. These
3                   ! constants are prologue dependent.
3                   !
3                   IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
3                   THEN
4                       BEGIN
4                       RRV_SIZE     = IRC$C_FIXOVHDSZ;
4                       DEL_RRV_SIZE = 2;
4                       END
3                   ELSE
4                       BEGIN
4                       RRV_SIZE     = IRC$C_FIXOVHSZ3;
4                       DEL_RRV_SIZE = 0;
3                       END;
3
3                   ! Delete/Squish the current primary data record's RRV and fix up
3                   ! the RRV bucket's freespace.
3                   !
4                   LENGTH = (.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
3                            - (.REC_ADDR + .RRV_SIZE);
3
3                   IF .LENGTH GTRU 0
3                   THEN
3                       RM$MOVE (.LENGTH,
3                                .REC_ADDR + .RRV_SIZE,
3                                .REC_ADDR + .DEL_RRV_SIZE);
3
3                   BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
3                                               - .RRV_SIZE + .DEL_RRV_SIZE;
3
3                   ! If the file is not a prologue 3 file, then the RRV of the current
3                   ! primary data record was just squished. The RRV pointer was
3                   ! removed, but the control byte and record ID fields remain. In
3                   ! this case RMS wants to setup the control byte of the squished RRV
3                   ! to indicate that it has been deleted, is an RRV, and doesn't
3                   ! contain a pointer.
3                   !
3                   IF .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3
3                   THEN
3                       REC_ADDR[IRC$B_CONTROL] = IRC$M_RRV OR IRC$M_DELETED
3                                                 OR IRC$M_NOPTRSZ;
2                   END;
2
2               ! Restore the address of the current primary data record and release the
2               ! RRV's bucket after marking it dirty.
2               !
2               REC_ADDR = .SAVE_REC_ADDR;
2
2               BDB[BDB$V_DRT] = 1;
2               RETURN RM$RLSBKT (RLS$M_WRT_THRU)
2
1               END;
```

```
                               2C   BB 00000 RM$DELETE_RRV::
                                                  PUSHR   #^M<R2,R3,R5>                        ; 0795
                 5E            08   C2 00002      SUBL2   #8, SP
                               7E   D4 00005      CLRL    -(SP)                                ; 0872
                            0000G   30 00007      BSBW    RM$KEY_DESC
                 5E            04   C0 0000A      ADDL2   #4, SP
                 6E            56   D0 0000D      MOVL    REC_ADDR, SAVE_REC_ADDR              ; 0874
                            0000G   30 00010      BSBW    RM$RECORD_ID                         ; 0878
                 5C            50   D0 00013      MOVL    R0, AP
                 55       18   A4   D0 00016      MOVL    24(BDB), BKT_ADDR                    ; 0885
                            0000G   30 0001A      BSBW    RM$FIND_BY_ID                        ; 0886
      05      07 A9            05   E1 0001D      BBC     #5, 7(IRAB), 1$                      ; 0892
                 66            20   88 00022      BISB2   #32, (REC_ADDR)                      ; 0894
                               53   11 00025      BRB     6$
      05      07 A9            06   E1 00027 1$:  BBC     #6, 7(IRAB), 2$                      ; 0900
                 66            20   8A 0002C      BICB2   #32, (REC_ADDR)                      ; 0902
                               49   11 0002F      BRB     6$
      03    00B7 CA            91      00031 2$:  CMPB    183(IFAB), #3                        ; 0912
                               09   1E 00036      BGEQU   3$
                 53            07   D0 00038      MOVL    #7, RRV_SIZE                         ; 0915
              04 AE            02   D0 0003B      MOVL    #2, DEL_RRV_SIZE                     ; 0916
                               06   11 0003F      BRB     4$                                   ; 0912
                 53            09   D0 00041 3$:  MOVL    #9, RRV_SIZE                         ; 0920
                          04   AE   D4 00044      CLRL    DEL_RRV_SIZE                         ; 0921
                 52       04   A5   3C 00047 4$:  MOVZWL  4(BKT_ADDR), R2                      ; 0927
      51         55            52   C1 0004B      ADDL3   R2, BKT_ADDR, R1
      50         56            53   C1 0004F      ADDL3   RRV_SIZE, REC_ADDR, R0               ; 0928
                 51            50   C2 00053      SUBL2   R0, LENGTH
                               0E   13 00056      BEQL    5$                                   ; 0930
                         04 BE46    9F 00058      PUSHAB  @DEL_RRV_SIZE[REC_ADDR]              ; 0934
                               50   DD 0005C      PUSHL   R0                                   ; 0933
                               51   DD 0005E      PUSHL   LENGTH                               ; 0932
                            0000G   30 00060      BSBW    RM$MOVE
                 5E            0C   C0 00063      ADDL2   #12, SP
      50         52            53   C3 00066 5$:  SUBL3   RRV_SIZE, R2, R0                     ; 0937
   04 A5         50       04   AE   A1 0006A      ADDW3   DEL_RRV_SIZE, R0, 4(BKT_ADDR)
      03    00B7 CA            91      00070      CMPB    183(IFAB), #3                        ; 0946
                               03   1E 00075      BGEQU   6$
                 66            1C   90 00077      MOVB    #28, (REC_ADDR)                      ; 0949
                 56            6E   D0 0007A 6$:  MOVL    SAVE_REC_ADDR, REC_ADDR              ; 0955
              0A A4            02   88 0007D      BISB2   #2, 10(BDB)                          ; 0957
                               02   DD 00081      PUSHL   #2                                   ; 0958
                            0000G   30 00083      BSBW    RM$RLSBKT
                 5E            0C   C0 00086      ADDL2   #12, SP                              ; 0960
                               2C   BA 00089      POPR    #^M<R2,R3,R5>
                               05      0008B      RSB

; Routine Size:  140 bytes,    Routine Base:  RM$RMS3 + 015A
```

```
1 %SBTTL  'RM$DELETE_SIDR'
1 GLOBAL ROUTINE RM$DELETE_SIDR : RL$RABREG_7 =
1
1 !++
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       This routine's responsibility is to position to the SIDR array element
1 !       pointing to the current primary data record for a given key of
1 !       reference and delete it. The secondary key in keybuffer 2, and the
1 !       RFA address of the current primary data record, found as part of the
1 !       local NRP context in the IRAB, are utilized in this positioning.
1 !       Deletion of the appropriate SIDR array element consists of one of the
1 !       following:
1 !
1 !       1. Removal of the entire SIDR if duplicates are not allowed.
1 !
1 !       2. Marking the SIDR array element as deleted and not recovering any
1 !          space if duplicates are allowed for this key of reference and the
1 !          file is a prologue 1 or 2 file.
1 !
1 !       3. Marking the SIDR array element as deleted and not recovering any
1 !          space if duplicates are allowed for this key of reference, the file
1 !          is a prologue 3 file, and the element is the first element in the SIDR
1 !          array.
1 !
1 !       4. Marking the SIDR element deleted and squishing out the space
1 !          occupied by the RRV pointer if duplicates are allowed for this key
1 !          of reference, the file is a prologue 3 file, and the element is not
1 !          the first element in the SIDR array.
1 !
1 !       5. Removal of the entire SIDR array if duplicates are allowed, this is
1 !          the first SIDR with this key value, the SIDR is not the physically
1 !          last SIDR in the bucket, and ever single element within the SIDR
1 !          array has been deleted.
1 !
1 ! CALLING SEQUENCE:
1 !
1 !       RM$DELETE_SIDR()
1 !
1 ! INPUT PARAMETERS:
1 !       NONE
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       IDX_DFN                 - address of index descriptor
1 !           IDX$B_KEYSZ         - size of alternate key
1 !
1 !       IRAB                    - address of IRAB
1 !           IRB$W_UDR_ID        - RFA VBN of the current primary data record
1 !           IRB$L_UDR_VBN       - RFA ID of the current primary data record
1 !
1 ! OUTPUT PARAMETERS:
1 !       NONE
1 !
1 ! IMPLICIT OUTPUTS:
1 !
```

```
!               IRAB                    - address of IRAB
!                       IRB$B_KEYSZ     - size of alternate key for key of reference
!                       IRB$B_STOPLEVEL - level of index to search to (set to 0)
!
! ROUTINE VALUE:
!
!               Status of the RM$RLSBKT call (success or error) that released
!                       the modified bucket.
!               BUG - if the SIDR array element could not be located
!
! SIDE EFFECTS:
!
!               Modified bucket is released.
!               IRB$V_POSDELETE set within IRB$W_SRCHFLGS.
!
!--

    BEGIN

    EXTERNAL REGISTER
        COMMON_RAB_STR,
        R_IDX_DFN_STR;

    GLOBAL REGISTER
        COMMON_IO_STR,
        R_REC_ADDR_STR;

    LABEL
        FIND_ELEMENT;

    LOCAL
        BEGIN_OF_SIDR;

    ! Since RMS is going to position so it can delete a SIDR array element,
    ! set the appropriate search flag, and make sure the key size is set up.
    !
    IRAB[IRB$B_STOPLEVEL] = 0;
    IRAB[IRB$W_SRCHFLAGS] = IRB$M_POSDELETE;
    IRAB[IRB$B_KEYSZ] = .IDX_DFN[IDX$B_KEYSZ];

    ! Position to the SIDR array element pointing to the current primary data
    ! record for this key of reference. This loop will only be exited either
    ! when the array element has been located or all SIDR elements with this
    ! key value are exhausted.
    !
FIND_ELEMENT:
    BEGIN

    LOCAL
        END_OF_SIDR,
        ID,
        STATUS,
        VBN;

    WHILE 1
    DO
        BEGIN
```

```
4
4              ! If RMS is unable to find an array element pointing to the current
4              ! primary data record, then something is very wrong. Return an internal
4              ! bug error, and save the status from RM$CSEARCH_TREE, in the RABs STV
4              ! field.
4              !
5              IF NOT (STATUS = RM$CSEARCH_TREE())
4              THEN
5                  BEGIN
5                  RAB[RAB$L_STV] = .STATUS;
5                  RETURN RMSERR(BUG);
4                  END;
4
4              ! Prepare to search the SIDR array for the element pointing to the
4              ! current primary data record.
4              !
4              BEGIN_OF_SIDR = .REC_ADDR;
4              END_OF_SIDR = RM$SIDR_END();
4
4              ! Position to the first array element in the SIDR array.
4
4              REC_ADDR = RM$SIDR_FIRST(0);
4
4              ! Search the current SIDR array for the element corresponding to the
4              ! current primary data record.
4              !
4              WHILE .REC_ADDR LSSA .END_OF_SIDR
4              DO
4
4                  ! If after extracting out the RFA pointer from the current SIDR
4                  ! array element, RMS finds that it does indeed point to the
4                  ! current primary data record, then exit the search loop
4                  !
4                  IF   RM$EXT_ARRY_RFA (VBN, ID)
4                        AND
5                       (.IRAB[IRB$W_UDR_ID] EQLU .ID)
4                        AND
5                       (.IRAB[IRB$L_UDR_VBN] EQLU .VBN)
4                  THEN
4                      LEAVE FIND_ELEMENT
4
4                  ! If the current array element is deleted or does not point to the
4                  ! current primary data record then proceed to the next element in
4                  ! the SIDR array.
4                  !
4                  ELSE
4                      RM$GETNXT_ARRAY();
3              END;
2          END;
2
2          ! Delete the SIDR array pointing to the current primary data record
2          ! for this key of reference. The deletion rules are stated above.
2          !
2          BDB = .IRAB[IRB$L_CURBDB];
2          IRAB[IRB$L_CURBDB] = 0;
2
2          BKT_ADDR = .BDB[BDB$L_ADDR];
```

```
; 1072      1132 2     RM$SQUISH_SIDR (0, .BEGIN_OF_SIDR);
; 1073      1133 2
; 1074      1134 2     ! Mark the bucket dirty, and release it.
; 1075      1135 2     !
; 1076      1136 2     BDB[BDB$V_DRT] = 1;
; 1077      1137 2     RETURN RM$RLSBKT(0);
; 1078      1138 2
; 1079      1139 1     END;


                       007C   8F  BB 00000 RM$DELETE_SIDR::
                                                   PUSHR   #^M<R2,R3,R4,R5,R6>                     ; 0962
                5E            08  C2 00004         SUBL2   #8, SP                                  ;
                        41    A9  94 00007         CLRB    65(IRAB)                                ; 1054
          42    A9            04  B0 0000A         MOVW    #4, 66(IRAB)                            ; 1055
        00A6    C9      20    A7  90 0000E         MOVB    32(IDX_DFN), 166(IRAB)                  ; 1056
                           0000G  30 00014 1$:     BSBW    RM$CSEARCH_TREE                         ; 1081
                54            50  D0 00017         MOVL    R0, STATUS                              ;
                0B            54  E8 0001A         BLBS    STATUS, 2$                              ;
          0C    A8            54  D0 0001D         MOVL    STATUS, 12(RAB)                         ; 1084
                50    8434    8F  3C 00021         MOVZWL  #33844, R0                              ; 1085
                              5E  11 00026         BRB     6$                                      ;
                53            56  D0 00028 2$:     MOVL    REC_ADDR, BEGIN_OF_SIDR                 ; 1091
                           0000G  30 0002B         BSBW    RM$SIDR_END                             ; 1092
                55            50  D0 0002E         MOVL    R0, END_OF_SIDR                         ;
                              7E  D4 00031         CLRL    -(SP)                                   ; 1096
                           0000G  30 00033         BSBW    RM$SIDR_FIRST                           ;
                5E            04  C0 00036         ADDL2   #4, SP                                  ;
                56            50  D0 00039         MOVL    R0, REC_ADDR                            ;
                55            56  D1 0003C 3$:     CMPL    REC_ADDR, END_OF_SIDR                   ; 1101
                              D3  1E 0003F         BGEQU   1$                                      ;
                              5E  DD 00041         PUSHL   SP                                      ; 1108
                        08    AE  9F 00043         PUSHAB  VBN                                     ;
                           0000G  30 00046         BSBW    RM$EXT_ARRY_RFA                         ;
                5E            08  C0 00049         ADDL2   #8, SP                                  ;
                11            50  E9 0004C         BLBC    R0, 4$                                  ;
 6E     00BC    C9    10      00  ED 0004F         CMPZV   #0, #16, 188(IRAB), ID                  ; 1110
                              08  12 00056         BNEQ    4$                                      ;
          04    AE    00B0    C9  D1 00058         CMPL    176(IRAB), VBN                          ; 1112
                              05  13 0005E         BEQL    5$                                      ;
                           0000G  30 00060 4$:     BSBW    RM$GETNXT_ARRAY                         ; 1121
                              D7  11 00063         BRB     3$                                      ; 1108
                54      20    A9  D0 00065 5$:     MOVL    32(IRAB), BDB                           ; 1128
                        20    A9  D4 00069         CLRL    32(IRAB)                                ; 1129
                55      18    A4  D0 0006C         MOVL    24(BDB), BKT_ADDR                       ; 1131
                              53  DD 00070         PUSHL   BEGIN_OF_SIDR                           ; 1132
                              7E  D4 00072         CLRL    -(SP)                                   ;
                           0000V  30 00074         BSBW    RM$SQUISH_SIDR                          ;
                5E            04  C0 00077         ADDL2   #4, SP                                  ;
          0A    A4            02  88 0007A         BISB2   #2, 10(BDB)                             ; 1136
                              6E  D4 0007E         CLRL    (SP)                                    ; 1137
                           0000G  30 00080         BSBW    RM$RLSBKT                               ;
                5E            04  C0 00083         ADDL2   #4, SP                                  ;
                5E            08  C0 00086 6$:     ADDL2   #8, SP                                  ; 1139
```

```
                        007C    8F  BA 00089          POPR    #^M<R2,R3,R4,R5,R6>                       ;
                                    05 0008D          RSB                                               ;

; Routine Size:  142 bytes,     Routine Base:  RM$RMS3 + 01E6
```

```
%SBTTL  'RMS$DELETE_UDR'
GLOBAL ROUTINE RMS$DELETE_UDR : RL$RABREG_4567 NOVALUE =

!++
!
! FUNCTIONAL DESCRIPTION:
!
!       This routine's responsibility is the deletion of a primary data record.
!       Most but not all of the time, the record being deleted is the current
!       primary data record. The rules for how primary data records are deleted
!       are as follows:
!
!       1. If the primary data record is marked deleted, then the entire record
!          is always deleted.
!
!       2. If duplicate primary keys are not allowed, and the record is not the
!          last primary data record in the bucket then the entire primary data
!          record is deleted.
!
!       3. If duplicate primary keys are not allowed, and the record is the
!          last primary data record in the bucket then the primary data record
!          is marked deleted, and the space occupied by the data portion of the
!          record is reclaimed if the file's prologue version is 3.
!
!       4. If duplicate primary keys are allowed then the primary data record
!          is marked deleted, and the space occupied by the data portion of the
!          record is recovered if the file's prologue version is 3.
!
!       5. If the state bit IRB$V_RU_DELETE is set, then the primary data
!          record is just marked RU_DELETE and no space is reclaimed.
!
!       6. If the state bit IRB$V_RU_UNDEL is set, then the primary data record
!          is un-deleted by clearing the RU_DELETE bit within the record control
!          byte.
!
!       7. If the primary data record is completely deleted, the record was in
!          its original bucket (ie - a RRV does not exist), and the file's
!          prologue version is 1 or 2, then a two-byte RRV is created at the
!          end of the bucket for this record to prevent its ID from being
!          recycled.
!
! CALLING SEQUENCE:
!
!       RMS$DELETE_UDR()
!
! INPUT PARAMETERS:
!       NONE
!
! IMPLICIT INPUTS:
!
!       BDB                     - address of BDB for primary data bucket buffer
!           BDB$L_ADDR          - address of primary data bucket buffer
!           BDB$L_VBN           - VBN of primary data bucket
!
!       IDX_DFN                 - address of index descriptor for primary key
!           IDX$V_DUPKEYS       - if set, duplicate primary keys are allowed
!           IDX$V_KEY_COMPR     - if set, primary key compression is enabled
```

```
!
!       IFAB                    - address of IFAB
!           IFB$W_KBUFSZ        - size of each keybuffer
!           IFB$B_PLG_VER       - prologue version of the file
!
!       IRAB                    - address of IRAB
!           IRB$L_KEYBUF        - address of the contigious keybuffers
!           IRB$V_RU_DELETE     - if set, mark RU_DELETE and do not reclaim
!           IRB$V_RU_UNDEL      - if set, un-delete the RRV
!
!       REC_ADDR                - address of primary data record to be deleted
!
! OUTPUT PARAMETERS:
!       NONE
!
! IMPLICIT OUTPUTS:
!       NONE
!
! ROUTINE VALUE:
!       NONE
!
! SIDE EFFECTS:
!
!       AP is trashed.
!       Keybuffer 5 is trashed (if the primary key of the following primary
!           data record had to be re-expanded).
!       The freespace offset in the bucket is updated to reflect the amount
!           of space reclaimed.
!       REC_ADDR is unchanged. It either points to the deleted record if the
!           target primary data record could not be completely removed, or
!           it points to whatever followed the deleted primary data record
!           (if anything) if it could.
!       If this is a prologue 1 or 2 file, and the primary data record which was
!           deleted is in its original bucket, then a two-byte RRV is created
!           to replace the deleted primary data record, provided the space
!           occupied by the record was completely recovered.
!
!--

    BEGIN

    BUILTIN
        AP;

    EXTERNAL REGISTER
        R_BDB_STR,
        COMMON_RAB_STR,
        R_IDX_DFN_STR,
        R_REC_ADDR_STR;

    GLOBAL REGISTER
        R_BKT_ADDR_STR;

    FIELD
        DELETE_FLAGS =
            SET
            BUILD_RRV           = [0,0,1,0],
```

```
                LAST_RECORD         = [0,1,1,0],
                RE_EXPAND_KEY       = [0,2,1,0]
                TES;

        LOCAL
            END_OF_BUCKET   : REF BBLOCK,
            FLAGS           : BLOCK[1,BYTE]
                              FIELD(DELETE_FLAGS),
            NEXT_REC_ADDR   : REF BBLOCK,
            REC_OVHD;

        ! If is is indicated that the primary data record should just be marked
        ! RU_DELETE and that no space should be reclaimed, then do so by setting
        ! the RU_DELETE bit within the RRV's control byte.
        !
        IF .IRAB[IRB$V_RU_DELETE]
        THEN
            BEGIN
            REC_ADDR[IRC$V_RU_DELETE] = 1;
            RETURN;
            END

        ! If it is indicated that the primary data record should be un-deleted,
        ! then do so by clearing the RU_DELETE bit in the pirmary data record's
        ! control byte.
        !
        ELSE
            IF .IRAB[IRB$V_RU_UNDEL]
            THEN
                BEGIN
                REC_ADDR[IRC$V_RU_DELETE] = 0;
                RETURN;
                END;

        ! Obtain the address of the primary data bucket, and compute the first
        ! free byte in the data bucket.
        !
        FLAGS = 0;
        BKT_ADDR = .BDB[BDB$L_ADDR];
        END_OF_BUCKET = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];

        ! Obtain the overhead for ALL records in this primary data bucket, and
        ! compute the address of the first primary data record which would follow
        ! the primary data record to be deleted.
        !
        BEGIN

        LOCAL
            REC_SIZE;

        REC_OVHD = RM$REC_OVHD(0; REC_SIZE);
        NEXT_REC_ADDR = .REC_ADDR + .REC_OVHD + .REC_SIZE;
        END;

        ! Determine whether the primary data record to be deleted is the last
        ! record in the bucket, and set the local state flag accordingly.
        !
```

```
3       IF  (.NEXT_REC_ADDR EQLA .END_OF_BUCKET)
2            OR
2           .NEXT_REC_ADDR[IRC$V_RRV]
2       THEN
2           FLAGS[LAST_RECORD] = 1;
2
2       ! If the target primary data record can not be completely deleted either
2       ! because duplicates primary keys are allowed or it is the last record
2       ! in the bucket, mark the record deleted, and squish out the data portion
2       ! of the primary data record if it is squishable.
2       !
2       IF  NOT .REC_ADDR[IRC$V_DELETED]
2            AND
3           (.IDX_DFN[IDX$V_DUPKEYS]
3                     OR
3                    .FLAGS[LAST_RECORD])
2       THEN
3           BEGIN
3           RM$SQUISH_DATA();
3           REC_ADDR[IRC$V_DELETED] = 1;
3           RETURN;
3           END
3
3       ! The primary data record can be completely deleted. It is either marked
3       ! deleted (the only reason why RMS would be calling this routine would be
3       ! to elimiate it entirely), or duplicates are not allowed and it is not
3       ! the last primary data record in the bucket.
3       !
2       ELSE
3           BEGIN
3
3           LOCAL
3               UDR_ID;
3
3           ! If the file is a prologue 1 or 2 file and the primary data record to
3           ! be deleted is in its original bucket (ie - there is no RRV for it),
3           ! then a two-byte RRV will have to be created for it at the end of the
3           ! bucket inorder to reserve its ID and prevent it from being recycled.
3           !
3           AP = 3;
3
4           IF (.IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
3               AND
4               (RM$RECORD_VBN() EQLA .BDB[BDB$L_VBN])
3           THEN
4               BEGIN
4               FLAGS[BUILD_RRV] = 1;
4               UDR_ID = .REC_ADDR[IRC$B_ID];
3               END;
3
3           ! If primary key compression is enabled, and this primary data record
3           ! is not the last record in the file, then the key of the following
3           ! record, whose front compression is based on this record, will have
3           ! to be re-expanded, after this target primary data record is
3           ! completely removed. Set the local state bit accordingly and save the
3           ! entire key portion (both control bytes and key) of the target primary
3           ! data record in keybuffer 5 to be used in re-expanded the key of the
```

```
        ! primary data record that follows.
        !
        IF .IDX_DFN[IDX$V_KEY_COMPR]
            AND
           NOT .FLAGS[LAST_RECORD]
        THEN
            BEGIN
            FLAGS[RE_EXPAND_KEY] = 1;

            RM$MOVE (.(.REC_ADDR + .REC_OVHD)<0,8> + 2,
                     .REC_ADDR + .REC_OVHD,
                     KEYBUF_ADDR(5));
            END;

        ! If the primary data record being deleted is not the last entity in
        ! the bucket, recover the space it occupies by shifting everything
        ! that follows, and update the freespace offset in the bucket
        ! accordingly. If the primary data record being deleted is the last
        ! entity in the primary data bucket the space it occupies maybe
        ! recovered by just adjusting the freespace offset.
        !
        IF (.NEXT_REC_ADDR LSSA .END_OF_BUCKET)
        THEN
            RM$MOVE (.END_OF_BUCKET - .NEXT_REC_ADDR,
                     .NEXT_REC_ADDR,
                     .REC_ADDR);

        BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
                                            - (.NEXT_REC_ADDR - .REC_ADDR);

        ! If there is a record following the primary data record just deleted,
        ! whose primary key is to be re-expanded, re-expand it. The routine
        ! RM$EXPAND_KEYD will take care of re-adjusting the bucket freespace
        ! offset.
        !
        IF .FLAGS[RE_EXPAND_KEY]
        THEN
            RM$EXPAND_KEYD (KEYBUF_ADDR(5), .REC_ADDR + .REC_OVHD);

        ! If a two-byte RRV must be built for the deleted primary data record,
        ! then build it at the end of the bucket, and adjust the bucket
        ! freespace offset to reflect the RRV's size.
        !
        IF .FLAGS[BUILD_RRV]
        THEN
            BEGIN
            END_OF_BUCKET = .BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE];
            END_OF_BUCKET[IRC$B_CONTROL] = IRC$M_DELETED OR IRC$M_NOPTRSZ
                                                        OR IRC$M_RRV;
            END_OF_BUCKET[IRC$B_ID] = .UDR_ID;
            BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE] + 2;
            END;

        END;

    END;
```

```
                                2C  BB 00000 RM$DELETE_UDR::
                                                   PUSHR   #^M<R2,R3,R5>                              ; 1141
                  5E            0C  C2 00002       SUBL2   #12, SP                                    ;
    05       07   A9            05  E1 00005       BBC     #5, 7(IRAB), 1$                            ; 1269
                  66            20  88 0000A       BISB2   #32, (REC_ADDR)                            ; 1272
                                48  11 0000D       BRB     6$                                         ; 1271
    05       07   A9            06  E1 0000F 1$:   BBC     #6, 7(IRAB), 2$                            ; 1281
                  66            20  8A 00014       BICB2   #32, (REC_ADDR)                            ; 1284
                                3E  11 00017       BRB     6$                                         ; 1283
                                53  94 00019 2$:   CLRB    FLAGS                                      ; 1291
                  55       18   A4  D0 0001B       MOVL    24(BDB), BKT_ADDR                          ; 1292
                  52       04   A5  3C 0001F       MOVZWL  4(BKT_ADDR), END_OF_BUCKET                 ; 1293
                  52            55  C0 00023       ADDL2   BKT_ADDR, END_OF_BUCKET                    ;
                                51  D4 00026       CLRL    R1                                         ; 1304
                             0000G  30 00028       BSBW    RM$REC_OVHD                                ;
             04   AE            50  D0 0002B       MOVL    R0, REC_OVHD                               ;
    50            56       04   AE  C1 0002F       ADDL3   REC_OVHD, REC_ADDR, R0                     ; 1305
    6E            50            51  C1 00034       ADDL3   REC_SIZE, R0, NEXT_REC_ADDR                ;
                  52            6E  D1 00038       CMPL    NEXT_REC_ADDR, END_OF_BUCKET               ; 1311
                                05  13 0003B       BEQL    3$                                         ;
    03       00   BE            03  E1 0003D       BBC     #3, @NEXT_REC_ADDR, 4$                     ; 1313
                  53            02  88 00042 3$:   BISB2   #2, FLAGS                                  ; 1315
    11            66            02  E0 00045 4$:   BBS     #2, (REC_ADDR), 7$                         ; 1322
                  04       1C   A7  E8 00049       BLBS    28(IDX_DFN), 5$                            ; 1324
    09            53            01  E1 0004D       BBC     #1, FLAGS, 7$                              ; 1326
                             0000V  30 00051 5$:   BSBW    RM$SQUISH_DATA                             ; 1329
                  66            04  88 00054       BISB2   #4, (REC_ADDR)                             ; 1330
                              008B  31 00057 6$:   BRW     12$                                        ; 1328
                  5C            03  D0 0005A 7$:   MOVL    #3, AP                                     ; 1350
                  03     00B7   CA  91 0005D       CMPB    183(IFAB), #3                              ; 1352
                                11  1E 00062       BGEQU   8$                                         ;
                             0000G  30 00064       BSBW    RM$RECORD_VBN                              ; 1354
             1C   A4            50  D1 00067       CMPL    R0, 28(BDB)                                ;
                                08  12 0006B       BNEQ    8$                                         ;
                  53            01  88 0006D       BISB2   #1, FLAGS                                  ; 1357
             08   AE       01   A6  9A 00070       MOVZBL  1(REC_ADDR), UDR_ID                        ; 1358
    22       1C   A7            06  E1 00075 8$:   BBC     #6, 28(IDX_DFN), 9$                        ; 1370
    1E            53            01  E0 0007A       BBS     #1, FLAGS, 9$                              ; 1372
                  53            04  88 0007E       BISB2   #4, FLAGS                                  ; 1375
                  50     00B4   CA  3C 00081       MOVZWL  180(IFAB), R0                              ; 1379
                          60 B940   DF 00086       PUSHAL  @96(IRAB)[R0]                              ;
                          08 BE46   9F 0008A       PUSHAB  @REC_OVHD[REC_ADDR]                        ; 1378
                  7E      0C BE46   9A 0008E       MOVZBL  @REC_OVHD[REC_ADDR], -(SP)                 ; 1377
                  6E            02  C0 00093       ADDL2   #2, (SP)                                   ;
                             0000G  30 00096       BSBW    RM$MOVE                                    ;
                  5E            0C  C0 00099       ADDL2   #12, SP                                    ;
                  52            6E  D1 0009C 9$:   CMPL    NEXT_REC_ADDR, END_OF_BUCKET               ; 1389
                                10  1E 0009F       BGEQU   10$                                        ;
                                56  DD 000A1       PUSHL   REC_ADDR                                   ; 1393
                           04   AE  DD 000A3       PUSHL   NEXT_REC_ADDR                              ; 1392
    7E            52       08   AE  C3 000A6       SUBL3   NEXT_REC_ADDR, END_OF_BUCKET, -(SP)        ; 1391
                             0000G  30 000AB       BSBW    RM$MOVE                                    ;
                  5E            0C  C0 000AE       ADDL2   #12, SP                                    ;
```

```
          50          56          6E  C3 000B1 10$:   SUBL3   NEXT_REC_ADDR, REC_ADDR, R0             ; 1396
                 04   A5          50  A0 000B5        ADDW2   R0, 4(BKT_ADDR)                         ;
          12          53          02  E1 000B9        BBC     #2, FLAGS, 11$                          ; 1403
          51          56    04    AE  C1 000BD        ADDL3   REC_OVHD, REC_ADDR, R1                  ; 1405
                      50  00B4    CA  3C 000C2        MOVZWL  180(IFAB), R0                           ;
                      50    60  B940  DE 000C7        MOVAL   @96(IRAB)[R0], R0                       ;
                              0000G   30 000CC        BSBW    RM$EXPAND_KEYD                          ;
                      13          53  E9 000CF 11$:   BLBC    FLAGS, 12$                              ; 1411
                      52    04    A5  3C 000D2        MOVZWL  4(BKT_ADDR), END_OF_BUCKET              ; 1414
                      52          55  C0 000D6        ADDL2   BKT_ADDR, END_OF_BUCKET                 ;
                      62          1C  90 000D9        MOVB    #28, (END_OF_BUCKET)                    ; 1416
                 01   A2    08    AE  90 000DC        MOVB    UDR_ID, 1(END_OF_BUCKET)                ; 1417
                 04   A5          02  A0 000E1        ADDW2   #2, 4(BKT_ADDR)                         ; 1418
                      5E          0C  C0 000E5 12$:   ADDL2   #12, SP                                 ; 1423
                                  2C  BA 000E8        POPR    #^M<R2,R3,R5>                           ;
                                      05 000EA        RSB                                             ;

; Routine Size:  235 bytes,    Routine Base:  RM$RMS3 + 0274
```

```
%SBTTL  'RM$SQUISH_DATA'
ROUTINE RM$SQUISH_DATA : RL$SQUISH_DATA NOVALUE =

!++
!
! FUNCTIONAL DESCRIPTION:
!
!       This routine's responsibility is the deletion of the data part of
!       the current primary data record. This deletion can only take place if
!       the file is a prologue 3 file.
!
! CALLING SEQUENCE:
!
!       RM$SQUISH_DATA()
!
! INPUT PARAMETERS:
!       NONE
!
! IMPLICIT INPUTS:
!
!       BKT_ADDR                        - address of the primary data bucket
!
!       IDX_DFN                         - address of the primary key index descriptor
!           IDX$V_KEY_COMP              - if set, key compression is enabled
!           IDX$B_KEYSZ                 - size of the key
!           IDX$V_REC_COMP              - if set, record compression is enabled
!
!       IFAB                            - address of the IFAB
!           IFB$B_PLG_VER               - prologue version of the file
!
!       REC_ADDR                        - address of the current primary data record
!
! OUTPUT PARAMETERS:
!       NONE
!
! IMPLICIT OUTPUTS:
!       NONE
!
! ROUTINE VALUE:
!       NONE
!
! SIDE EFFECTS:
!
!       The freespace in the bucket is updated to reflect the space reclaimed.
!
!--
    BEGIN

    EXTERNAL REGISTER
        R_BKT_ADDR_STR,
        R_IDX_DFN_STR,
        R_IFAB_STR,
        R_REC_ADDR_STR;

    GLOBAL REGISTER
        R_RAB,
        R_IRAB,
```

```
        R_IMPURE,
        R_BDB;

LOCAL
        REC_SIZE,
        KEY_SIZE,
        REC_OVHD,
        SIZE;

! If this is not a prologue 3 file then nothing can be done; however, if
! this is a prologue 3 file then as the primary key is always kept separate
! from the data portion of a prologue 3 data record, the data portion
! of the current primary data record can always be squished out, and its
! space recovered.
!
IF .IFAB[IFB$B_PLG_VER] NEQ PLG$C_VER_3
THEN
        RETURN;

! Obtain the size of the record overhead and the size of the current
! primary data record. Note that the size of the key (and any key specific
! control bytes) is always included as part of the size of the current
! primary data record.
!
REC_OVHD = RM$REC_OVHD(0; REC_SIZE);

! Compute the contribution of the primary key of the record to the size of
! the current primary data record. If primary key compression is enabled,
! then the key size will include the two bytes of key compression overhead.
!
IF .IDX_DFN[IDX$V_KEY_COMPR]
THEN
        KEY_SIZE = .(.REC_ADDR + .REC_OVHD)<0,8> + 2
ELSE
        KEY_SIZE = .IDX_DFN[IDX$B_KEYSZ];

! Compute the size of the data portion of the current primary data record.
! If the current primary data record consists of the primary key alone,
! return, as there is no data portion to squish out.
!
IF ((SIZE = .REC_SIZE - .KEY_SIZE) EQLU 0)
THEN
        RETURN;

! Squish out the data portion of the current primary data record.
!
RM$MOVE ((.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
                        - (.REC_ADDR + .REC_OVHD + .REC_SIZE),
             .REC_ADDR + .REC_OVHD + .REC_SIZE,
             .REC_ADDR + .REC_OVHD + .KEY_SIZE);

! Update the record size of the current primary data record to reflect
! the squishing out of the data portion of the record. NOTE that if the
! record is fixed length and both key and record compression are disabled,
! then there will be no record size field to update.
!
IF NOT (.IFAB[IFB$B_RFMORG] EQLU FAB$C_FIX
```

```
; 1480      1538  3                 AND
; 1481      1539  3               NOT .IDX_DFN[IDX$V_KEY_COMPR]
; 1482      1540  3                 AND
; 1483      1541  3               NOT .IDX_DFN[IDX$V_REC_COMPR])
; 1484      1542  2         THEN
; 1485      1543  2             (.REC_ADDR + .REC_OVHD - 2)<0,16> = .(.REC_ADDR + .REC_OVHD - 2)<0,16>
; 1486      1544  2                                                  - .SIZE;
; 1487      1545  2
; 1488      1546  2         ! Update the freespace pointer in the bucket to reflect the space that
; 1489      1547  2         ! has been recovered by the squishing out of the data portion of the
; 1490      1548  2         ! current primary data record.
; 1491      1549  2         !
; 1492      1550  2         BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE] - .SIZE;
; 1493      1551  2
; 1494      1552  1         END;



                              0B1C    8F    BB 00000 RM$SQUISH_DATA:
                                                             PUSHR    #^M<R2,R3,R4,R8,R9,R11>                  ; 1425
                   03         00B7    CA    91 00004         CMPB     183(IFAB), #3                            ; 1496
                                      58    12 00009         BNEQ     5$
                                      51    D4 0000B         CLRL     R1                                       ; 1505
                                   0000G    30 0000D         BSBW     RM$REC_OVHD
                   52                 50    D0 00010         MOVL     R0, REC_OVHD
          09           1C     A7      06    E1 00013         BBC      #6, 28(IDX_DFN), 1$                      ; 1511
                       54           6246    9A 00018         MOVZBL   (REC_OVHD)[REC_ADDR], KEY_SIZE           ; 1513
                       54             02    C0 0001C         ADDL2    #2, KEY_SIZE
                                      04    11 0001F         BRB      2$
                       54     20      A7    9A 00021 1$:     MOVZBL   32(IDX_DFN), KEY_SIZE                    ; 1515
          53           51             54    C3 00025 2$:     SUBL3    KEY_SIZE, REC_SIZE, SIZE                 ; 1521
                                      38    13 00029         BEQL     5$
          50           56             52    C1 0002B         ADDL3    REC_OVHD, REC_ADDR, R0                   ; 1530
                                    6440    9F 0002F         PUSHAB   (KEY_SIZE)[R0]
                       51             50    C0 00032         ADDL2    R0, R1                                   ; 1529
                                      51    DD 00035         PUSHL    R1
                       50     04      A5    3C 00037         MOVZWL   4(BKT_ADDR), R0                          ; 1527
                       50             55    C0 0003B         ADDL2    BKT_ADDR, R0
          7E           50             51    C3 0003E         SUBL3    R1, R0, -(SP)                            ; 1528
                                   0000G    30 00042         BSBW     RM$MOVE
                       5E             0C    C0 00045         ADDL2    #12, SP
                       01     50      AA    91 00048         CMPB     80(IFAB), #1                             ; 1537
                                      0A    12 0004C         BNEQ     3$
          05           1C     A7      06    E0 0004E         BBS      #6, 28(IDX_DFN), 3$                      ; 1539
                              1C      A7    95 00053         TSTB     28(IDX_DFN)                              ; 1541
                                      07    18 00056         BGEQ     4$
                              FE A246       9F 00058 3$:     PUSHAB   -2(REC_OVHD)[REC_ADDR]                   ; 1544
                       9E             53    A2 0005C         SUBW2    SIZE, @(SP)+
                 04    A5             53    A2 0005F 4$:     SUBW2    SIZE, 4(BKT_ADDR)                        ; 1550
                              0B1C    8F    BA 00063 5$:     POPR     #^M<R2,R3,R4,R8,R9,R11>                  ; 1552
                                            05 00067         RSB

; Routine Size:  104 bytes,    Routine Base:  RM$RMS3 + 035F
```

```
1 %SBTTL  'RM$SQUISH_SIDR'
1 GLOBAL ROUTINE RM$SQUISH_SIDR (SCAN, BEGIN_OF_SIDR) : RL$RABREG_567 =
1
1 !++
1 !
1 ! FUNCTIONAL DESCRIPTION:
1 !
1 !       This routine's responsibility is to delete the SIDR array element
1 !       pointing to the current primary data record for this key of reference.
1 !       Deletion of the SIDR array element goes according to one of the
1 !       following rules:
1 !
1 !       1. Removal of the entire SIDR if duplicates are not allowed. NOTE that
1 !          if the input parameter SCAN is 1 and the file is a prologue 3 file
1 !          then for the purpose of this SIDR deletion it is assumed that this
1 !          key of reference does allow duplicates (See rules 2 through 5).
1 !
1 !       2. Marking the SIDR array element as deleted and not recovering any
1 !          space if duplicates are allowed for this key of reference and the
1 !          file is a prologue 1 or 2 file.
1 !
1 !       3. Marking the SIDR array element as deleted and not recovering any
1 !          space if duplicates are allowed for this key of reference, the file
1 !          is a prologue 3 file, and the element is the first element in the
1 !          SIDR array.
1 !
1 !       4. Marking the SIDR element deleted and squishing out the space
1 !          occupied by the RRV pointer if duplicates are allowed for this key
1 !          of reference, the file is a prologue 3 file, and the element is not
1 !          the first element in the SIDR array.
1 !
1 !       5. Removal of the entire SIDR array if duplicates are allowed, this is
1 !          the first SIDR with this key value, the SIDR is not the physically
1 !          last SIDR in the bucket, and ever single element within the SIDR
1 !          array has been deleted.
1 !
1 !       6. If the state bit IRB$V_RU_DELETE is set, then the SIDR array element
1 !          is just marked RU_DELETE and no space is reclaimed.
1 !
1 !       7. If the state bit IRB$V_RU_UNDEL is set, then the SIDR array element
1 !          is un-deleted by clearing the RU_DELETE bit within the element's
1 !          control byte.
1 !
1 ! CALLING SEQUENCE:
1 !
1 !       BSBW RM$SQUISH_SIDR()
1 !
1 ! INPUT PARAMETERS:
1 !
1 !       SCAN            - if 1, scan the current SIDR array (if Prologue 3 file)
1 !
1 !       BEGIN_OF_SIDR   - pointer to the beginning of the SIDR record
1 !
1 ! IMPLICIT INPUTS:
1 !
1 !       BKT_ADDR                - address of the SIDR bucket
1 !
```

```
!       IDX_DFN                 - address of the index descriptor
!           IDX$V_DUPKEYS       - if set, duplicate keys are allowed
!           IDX$V_KEY_COMPR     - if set, SIDR key compression is enabled
!
!       IFAB                    - address of IFAB
!           IFB$W_KBUFSZ        - size of one of the contigious keybuffers
!           IFB$B_PLG_VER       - prologue version of file
!
!       IRAB                    - address of IRAB
!           IRB$L_KEYBUF        - address of the contigious keybuffers
!           IRB$V_RU_DELETE     - if set, mark RU_DELETE and do not reclaim
!           IRB$V_RU_UNDEL      - if set, un-delete the RRV
!
!       REC_ADDR                - address of the SIDR array element
!
! OUTPUT PARAMETERS:
!       NONE
!
! IMPLICIT OUTPUTS:
!
!       REC_ADDR        - address of next SIDR if the entire SIDR was deleted
!                         otherwise unchanged.
!
! ROUTINE VALUE:
!
!       1               - some space was recovered.
!       0               - no space was recovered.
!
! SIDE EFFECTS:
!
!       Keybuffer 5 will have been trashed, if any key re-expansion occurred.
!       The freespace in the bucket is updated to reflect the space reclaimed.
!       If the SIDR is completely deleted, SIDR key compression is enabled, and
!           a SIDR follows the completely deleted SIDR, then the key of this
!           following SIDR will have been re-expanded.
!
!--

    BEGIN

    EXTERNAL REGISTER
        R_BKT_ADDR_STR,
        COMMON_RAB_STR,
        R_IDX_DFN_STR,
        R_REC_ADDR_STR;

    LABEL
        DUPS;

    LOCAL
        DELETE_START,
        DELETE_END,
        FLAGS           : BLOCK[1],
        LENGTH,
        NEXT_REC_ADDR,
        RECORD_OVHD,
        SAVE_REC_ADDR   : REF BBLOCK;
```

```
2
2       MAP
2           BEGIN_OF_SIDR   : REF BBLOCK;
2
2       MACRO
2           DELETE_SIDR       = 0,0,1,0 %,
2           SQUISH_SIDR       = 0,1,1,0 %,
2           RE_EXPAND_KEY     = 0,2,1,0 %;
2
2       ! If is is indicated that the SIDR array element should just be marked
2       ! RU_DELETE and that no space should be reclaimed, then do so by setting
2       ! the RU_DELETE bit within the element's control byte.
2       !
2       IF .IRAB[IRB$V_RU_DELETE]
2       THEN
3           BEGIN
3           REC_ADDR[IRC$V_RU_DELETE] = 1;
3           RETURN 0;
3           END
3
3       ! If it is indicated that the SIDR array element should be un-deleted,
3       ! then do so by clearing the RU_DELETE bit in the element's control byte.
3       !
2       ELSE
2           IF .IRAB[IRB$V_RU_UNDEL]
2           THEN
3               BEGIN
3               REC_ADDR[IRC$V_RU_DELETE] = 0;
3               RETURN 0;
2               END;
2
2       ! Save the address of the current SIDR element, and zero out the local
2       ! flag field.
2       !
2       FLAGS = 0;
2       SAVE_REC_ADDR = .REC_ADDR;
2
2       ! Determine the address of the first byte past the end of the current
2       ! SIDR.
2       !
3       BEGIN
3
3       LOCAL
3           REC_SIZE;
3
3       REC_ADDR = .BEGIN_OF_SIDR;
3       RECORD_OVHD = RM$REC_OVHD(-1; REC_SIZE);
3       NEXT_REC_ADDR = .REC_ADDR + .RECORD_OVHD + .REC_SIZE;
2       END;
2
2       ! If this secondary key of reference does not allow duplicate key values
2       ! and either the file's prologue version is 1 or 2; or, the input parameter
2       ! SCAN is 0, then the entire SIDR maybe deleted.
2       !
2       IF  NOT .IDX_DFN[IDX$V_DUPKEYS]
2             AND
3           (NOT .SCAN
```

```
3               OR
3               .IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
2       THEN
2           FLAGS[DELETE_SIDR] = 1
2
2       ! If this key of reference does allow duplicate SIDR keys or duplicates are
2       ! not allowed but the file's prologue version is 3 and a scan of then entire
2       ! SIDR array has been requested (SCAN is set to 1), then mark the current
2       ! element as deleted and under certain circumstances, reclaim the space
2       ! occupied by the SIDR array element's RRV pointer. Under very restricted
2       ! circumstances it will also be possible to reclaim the space occupied by
2       ! the entire SIDR.
2       !
2       ELSE
2   DUPS:
3           BEGIN
3           SAVE_REC_ADDR[IRC$V_DELETED] = 1;
3
3           ! If the file is a prologue 2 file then marking the element deleted is
3           ! all that can be done.
3           !
4           IF (.IFAB[IFB$B_PLG_VER] LSSU PLG$C_VER_3)
3           THEN
4               BEGIN
4               REC_ADDR = .SAVE_REC_ADDR;
4               RETURN 0;
4               END
4
4           ! The file is a prologue 3 file. If every single array element in this
4           ! SIDR array is deleted, if the SIDR is not physically the last SIDR in
4           ! the bucket (this restriction applies to duplicates keys allowed only)
4           ! and if this SIDR is the first such SIDR with this key value in the
4           ! file then it will be possible to delete the entire SIDR; otherwise,
4           ! the space occupied by the element's RRV pointer is reclaimed unless
4           ! it is the first element in the array in which case nothing more can
4           ! be done.
4           !
3           ELSE
4               BEGIN
4
4               LABEL
4                   ENTIRE_SIDR;
4
4               LOCAL
4                   FIRST_SIDR      : REF BBLOCK;
4
4               ! Obtain the address of the first array element in the SIDR array.
4               !
4               FIRST_SIDR = RM$SIDR_FIRST(0);
4
4               ! If the first element in the array (which maybe the element being
4               ! deleted) is marked deleted, and this SIDR is the first such
4               ! record in the file with this key value, then it still maybe
4               ! possible to delete the entire SIDR.
4               !
4               IF  .FIRST_SIDR[IRC$V_DELETED]
4                   AND
```

```
                                .FIRST_SIDR[IRC$V_FIRST_KEY]
                            THEN
    ENTIRE_SIDR:            BEGIN

                            LOCAL
                                SCAN_START;

                            ! If the current SIDR is physically the last SIDR in the bucket
                            ! and duplicates keys are allowed then it will not be possible
                            ! to reclaim the space occupied by the entire SIDR even if all
                            ! its elements are deleted.
                            !
                            IF .NEXT_REC_ADDR GEQA (.BKT_ADDR + .BKT_ADDR[BKT$W_FREESPACE])
                                AND
                               .IDX_DFN[IDX$V_DUPKEYS]
                            THEN
                                LEAVE ENTIRE_SIDR;

                            ! Scan the SIDR array starting with the second element up to
                            ! but not including the target element making sure that all
                            ! these elements have been deleted. If a live element is found
                            ! then the space occupied by the entire SIDR can not be
                            ! reclaimed.
                            !
                            SCAN_START = .FIRST_SIDR + .FIRST_SIDR[IRC$V_PTRSZ]
                                                     + IRC$C_DATPTRBS3
                                                     + 1;

                            IF (.SCAN_START LSSA .SAVE_REC_ADDR)
                            THEN
                                IF NOT CH$FAIL (CH$FIND_NOT_CH
                                                (.SAVE_REC_ADDR - .SCAN_START,
                                                 .SCAN_START,
                                                 %CHAR(IRC$M_DELETED)
                                                         OR
                                                        %CHAR(IRC$M_NOPTRSZ)))
                                THEN
                                    LEAVE ENTIRE_SIDR;

                            ! Scan the SIDR array starting with the first element past the
                            ! target element and ending with the last element in the SIDR
                            ! making sure that all these elements have been deleted. If a
                            ! live element is found then the space occupied by the entire
                            ! SIDR can not be reclaimed.
                            !
                            SCAN_START = .SAVE_REC_ADDR + .SAVE_REC_ADDR[IRC$V_PTRSZ]
                                                        + IRC$C_DATPTRBS3
                                                        + 1;

                            IF (.SCAN_START LSSA .NEXT_REC_ADDR)
                            THEN
                                IF NOT CH$FAIL (CH$FIND_NOT_CH
                                                (.NEXT_REC_ADDR - .SCAN_START,
                                                 .SCAN_START,
                                                 %CHAR(IRC$M_DELETED)
                                                         OR
                                                        %CHAR(IRC$M_NOPTRSZ)))
```

```
                        THEN
                            LEAVE ENTIRE_SIDR;

                    ! Every single element in the current SIDR has been found to be
                    ! deleted, so the space occupied by the entire SIDR maybe
                    ! reclaimed.
                    !
                    FLAGS[DELETE_SIDR] = 1;
                    LEAVE DUPS;
                    END;

                ! If it is not possible to delete the entire SIDR then set up to
                ! reclaim the space occupied by the element's RRV pointer unless the
                ! element is the first element in the array in which case nothing
                ! more can be done.
                !
                REC_ADDR = .SAVE_REC_ADDR;

                IF (.REC_ADDR EQLA .FIRST_SIDR)
                THEN
                    RETURN 0
                ELSE
                    FLAGS[SQUISH_SIDR] = 1;
                END;
            END;

        ! If the space occupies by the entire SIDR is to be reclaimed, set up to
        ! recover it.
        !
        IF .FLAGS[DELETE_SIDR]
        THEN
            BEGIN
            DELETE_START = .BEGIN_OF_SIDR;
            DELETE_END = .NEXT_REC_ADDR;

            ! If key compression is enabled, and this SIDR is not the last SIDR
            ! in the bucket, save the key of the current SIDR in keybuffer 5,
            ! so that it maybe used in expanding the key of the following
            ! record.
            !
            IF .IDX_DFN[IDX$V_KEY_COMPR]
            THEN
                BEGIN

                GLOBAL REGISTER
                    R_BDB;

                FLAGS[RE_EXPAND_KEY] = 1;

                RM$MOVE (.(.REC_ADDR + .RECORD_OVHD)<0,8> + 2,
                         .REC_ADDR + .RECORD_OVHD,
                         KEYBUF_ADDR(5));
                END;
            END

        ! If the space occupies by the RRV pointer is to be reclaimed, set up to
        ! recover it.
```

```
!
ELSE
    BEGIN

    DELETE_START = .REC_ADDR + 1;
    DELETE_END   = .DELETE_START + .REC_ADDR[IRC$V_PTRSZ]
                                 + IRC$C_DATPTRBS3;

    REC_ADDR[IRC$V_NOPTRSZ] = 1;
    REC_ADDR[IRC$V_PTRSZ]   = 0;

    ! Update the SIDR size field. As it is currently written, this
    ! updating assumes that the size field is the first two bytes
    ! (and the only two bytes) of the record overhead field.
    !
    (.BEGIN_OF_SIDR)<0,16> = .(.BEGIN_OF_SIDR)<0,16>
                                        - (.DELETE_END - .DELETE_START);
    END;

! Recover the space that can be recovered, and update the freespace offset
! in the SIDR bucket. If the SIDR is being completely deleted, and it is the
! last SIDR in the bucket then there will be nothing to move and only the
! bucket's freespace offset need be updated.
!
LENGTH = .BKT_ADDR[BKT$W_FREESPACE] - (.DELETE_END - .BKT_ADDR);

IF .LENGTH GTRU 0
THEN
    BEGIN

    GLOBAL REGISTER
        R_BDB;

    RM$MOVE (.LENGTH, .DELETE_END, .DELETE_START);
    END;

BKT_ADDR[BKT$W_FREESPACE] = .BKT_ADDR[BKT$W_FREESPACE]
                            - (.DELETE_END - .DELETE_START);

! If key compression is enabled, the space occupied by the current SIDR was
! completely reclaimed, and a SIDR follows whose key needs to be
! re-expanded, do so at this point.
!
IF .FLAGS[RE_EXPAND_KEY]
THEN
    RM$EXPAND_KEYD (KEYBUF_ADDR(5), .REC_ADDR + .RECORD_OVHD);

! Return indicating that some space has been recovered.
!
RETURN 1;
END;
```

```
1C  BB 00000 RM$SQUISH_SIDR::
```

```
                                                 PUSHR   #^M<R2,R3,R4>                        ; 1554
                      5E       0C  C2 00002      SUBL2   #12, SP                              ;
          05    07    A9       05  E1 00005      BBC     #5, 7(IRAB), 1$                      ; 1680
                      66       20  88 0000A      BISB2   #32, (REC_ADDR)                      ; 1683
                               43  11 0000D      BRB     4$                                   ; 1684
          05    07    A9       06  E1 0000F 1$:  BBC     #6, 7(IRAB), 2$                      ; 1691
                      66       20  8A 00014      BICB2   #32, (REC_ADDR)                      ; 1694
                               39  11 00017      BRB     4$                                   ; 1695
                    04     AE      D4 00019 2$:  CLRL    FLAGS                                ; 1701
                      54       56  D0 0001C      MOVL    REC_ADDR, SAVE_REC_ADDR              ; 1702
                      56    20 AE  D0 0001F      MOVL    BEGIN_OF_SIDR, REC_ADDR              ; 1712
                      51       01  CE 00023      MNEGL   #1, RT                               ; 1713
                            0000G  30 00026      BSBW    RM$REC_OVHD                          ;
                08    AE       50  D0 00029      MOVL    R0, RECORD_OVHD                      ;
          50          56    08 AE  C1 0002D      ADDL3   RECORD_OVHD, REC_ADDR, R0            ; 1714
          53          50       51  C1 00032      ADDL3   REC_SIZE, R0, NEXT_REC_ADDR          ;
                      0B    1C A7  E8 00036      BLBS    28(IDX_DFN), 3$                      ; 1721
                      7E    1C AE  E9 0003A      BLBC    SCAN, 10$                            ; 1723
                      03  00B7 CA  91 0003E      CMPB    183(IFAB), #3                        ; 1725
                               77  1F 00043      BLSSU   10$                                  ;
                      64       04  88 00045 3$:  BISB2   #4, (SAVE_REC_ADDR)                  ; 1740
                      03  00B7 CA  91 00048      CMPB    183(IFAB), #3                        ; 1745
                               06  1E 0004D      BGEQU   5$                                   ;
                      56       54  D0 0004F      MOVL    SAVE_REC_ADDR, REC_ADDR              ; 1748
                             0100  31 00052 4$:  BRW     17$                                  ; 1749
                               7E  D4 00055 5$:  CLRL    -(SP)                                ; 1772
                            0000G  30 00057      BSBW    RM$SIDR_FIRST                        ;
                      5E       04  C0 0005A      ADDL2   #4, SP                               ;
                      6E       50  D0 0005D      MOVL    R0, FIRST_SIDR                       ;
          5D    00    BE       02  E1 00060      BBC     #2, @FIRST_SIDR, 11$                 ; 1779
                    00     BE      95 00065      TSTB    @FIRST_SIDR                          ; 1781
                               58  18 00068      BGEQ    11$                                  ;
          50          04    A5     3C 0006A      MOVZWL  4(BKT_ADDR), R0                      ; 1793
          50                   55  C0 0006E      ADDL2   BKT_ADDR, R0                         ;
          50                   53  D1 00071      CMPL    NEXT_REC_ADDR, R0                    ;
                               04  1F 00074      BLSSU   6$                                   ;
                      48    1C A7  E8 00076      BLBS    28(IDX_DFN), 11$                     ; 1795
    50          00    BE    02 00  EF 0007A 6$:  EXTZV   #0, #2, @FIRST_SIDR, R0              ; 1805
                      51       6E  D0 00080      MOVL    FIRST_SIDR, R1                       ; 1807
                      52  05 A041  9E 00083      MOVAB   5(R0)[R1], SCAN_START                ;
                      54       52  D1 00088      CMPL    SCAN_START, SAVE_REC_ADDR            ; 1809
                               10  1E 0008B      BGEQU   8$                                   ;
          50          54       52  C3 0008D      SUBL3   SCAN_START, SAVE_REC_ADDR, R0        ; 1812
          62          50       14  3B 00091      SKPC    #20, R0, (SCAN_START)                ; 1815
                               02  12 00095      BNEQ    7$                                   ;
                               51  D4 00097      CLRL    R1                                   ;
                               51  D5 00099 7$:  TSTL    R1                                   ; 1816
                               25  12 0009B      BNEQ    11$                                  ;
    50          64          02 00  EF 0009D 8$:  EXTZV   #0, #2, (SAVE_REC_ADDR), R0          ; 1826
                      52  05 A044  9E 000A2      MOVAB   5(R0)[SAVE_REC_ADDR], SCAN_START     ; 1828
                      53       52  D1 000A7      CMPL    SCAN_START, NEXT_REC_ADDR            ; 1830
                               10  1E 000AA      BGEQU   10$                                  ;
          50          53       52  C3 000AC      SUBL3   SCAN_START, NEXT_REC_ADDR, R0        ; 1833
          62          50       14  3B 000B0      SKPC    #20, R0, (SCAN_START)                ; 1836
                               02  12 000B4      BNEQ    9$                                   ;
                               51  D4 000B6      CLRL    R1                                   ;
                               51  D5 000B8 9$:  TSTL    R1                                   ; 1837
```

```
                                06 12 000BA        BNEQ    11$                                  ;
              04   AE           01 88 000BC 10$:   BISB2   #1, FLAGS                            ; 1845
                                0C 11 000C0        BRB     12$                                  ; 1846
                   56           54 D0 000C2 11$:   MOVL    SAVE_REC_ADDR, REC_ADDR              ; 1854
                   6E           56 D1 000C5        CMPL    REC_ADDR, FIRST_SIDR                 ; 1856
                                88 13 000C8        BEQL    4$                                   ;
              04   AE           02 88 000CA        BISB2   #2, FLAGS                            ; 1860
                   2D     04 AE E9 000CE 12$:      BLBC    FLAGS, 13$                           ; 1867
                   51     20 AE D0 000D2           MOVL    BEGIN_OF_SIDR, DELETE_START          ; 1870
                   52           53 D0 000D6        MOVL    NEXT_REC_ADDR, DELETE_END            ; 1871
         3D   1C   A7           06 E1 000D9        BBC     #6, 28(IDX_DFN), 14$                 ; 1878
              04   AE           04 88 000DE        BISB2   #4, FLAGS                            ; 1885
                   50   00B4 CA 3C 000E2           MOVZWL  180(IFAB), R0                        ; 1889
                        60 B940 DF 000E7           PUSHAL  @96(IRAB)[R0]                        ;
                        0C BE46 9F 000EB           PUSHAB  @RECORD_OVHD[REC_ADDR]               ; 1888
                   7E   10 BE46 9A 000EF           MOVZBL  @RECORD_OVHD[REC_ADDR], -(SP)        ; 1887
                   6E           02 C0 000F4        ADDL2   #2, (SP)                             ;
                           0000G 30 000F7          BSBW    RM$MOVE                              ;
                   5E           0C C0 000FA        ADDL2   #12, SP                              ;
                                1C 11 000FD        BRB     14$                                  ; 1867
                   51     01 A6 9E 000FF 13$:      MOVAB   1(R6), DELETE_START                  ; 1899
    50   66        02           00 EF 00103        EXTZV   #0, #2, (REC_ADDR), R0               ; 1900
                   52   04 A041 9E 00108           MOVAB   4(R0)[DELETE_START], DELETE_END      ; 1901
                   66           10 88 0010D        BISB2   #16, (REC_ADDR)                      ; 1903
                   66           03 8A 00110        BICB2   #3, (REC_ADDR)                       ; 1904
         50        51           52 C3 00113        SUBL3   DELETE_END, DELETE_START, R0         ; 1911
              20   BE           50 A0 00117        ADDW2   R0, @BEGIN_OF_SIDR                   ;
         50        55           52 C3 0011B 14$:   SUBL3   DELETE_END, BKT_ADDR, R0             ; 1919
                   53     04 A5 3C 0011F           MOVZWL  4(BKT_ADDR), R3                      ;
                   50           53 C0 00123        ADDL2   R3, LENGTH                           ;
                                0A 13 00126        BEQL    15$                                  ; 1921
                                51 DD 00128        PUSHL   DELETE_START                         ; 1928
                                05 BB 0012A        PUSHR   #^M<R0,R2>                           ;
                           0000G 30 0012C          BSBW    RM$MOVE                              ;
                   5E           0C C0 0012F        ADDL2   #12, SP                              ;
                   51           52 C2 00132 15$:   SUBL2   DELETE_END, R1                       ; 1932
              04   A5           51 A0 00135        ADDW2   R1, 4(BKT_ADDR)                      ;
         12   04   AE           02 E1 00139        BBC     #2, FLAGS, 16$                       ; 1938
         51        56     08 AE C1 0013E           ADDL3   RECORD_OVHD, REC_ADDR, R1            ; 1940
                   50   00B4 CA 3C 00143           MOVZWL  180(IFAB), R0                        ;
                   50   60 B940 DE 00148           MOVAL   @96(IRAB)[R0], R0                    ;
                           0000G 30 0014D          BSBW    RM$EXPAND_KEYD                       ;
                   50           01 D0 00150 16$:   MOVL    #1, R0                               ; 1944
                                02 11 00153        BRB     18$                                  ;
                                50 D4 00155 17$:   CLRL    R0                                   ; 1945
                   5E           0C C0 00157 18$:   ADDL2   #12, SP                              ;
                                1C BA 0015A        POPR    #^M<R2,R3,R4>                        ;
                                   05 0015C        RSB                                          ;

; Routine Size:  349 bytes,    Routine Base:  RM$RMS3 + 03C7


; 1889          1946  1
; 1890          1947  1 END
; 1891          1948  1
; 1892          1949  0 ELUDOM
```

```
;                                PSECT SUMMARY
;
;       Name                    Bytes                          Attributes
;
;   RM$RMS3                          1316  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       92         2          154         00:00.4



;                                COMMAND QUALIFIERS

;       BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS$:RM3DELETE/OBJ=OBJ$:RM3DELETE MSRC$:RM3DELETE/UPDATE=(ENH$:RM3DELETE)

; 1893          1950  0
; Size:         1316 code + 0 data bytes
; Run Time:        00:33.6
; Elapsed Time:    01:00.1
; Lines/CPU Min:     3483
; Lexemes/CPU-Min: 15181
; Memory Used:  163 pages
; Compilation Complete
```

0324 AH-BT13A-SE DIGITAL EQUIPMENT CORPORATION
VAX/VMS V4.0 CONFIDENTIAL AND PROPRIETARY

RM3FACE
LIS

RM3DISCON
LIS

RM3CONN
LIS

RM3DELETE
LIS

RM3CREATE
LIS