```
-- DiskKD.Mesa  Edited by Sandman on May 12, 1978  2:09 PM

DIRECTORY
  AllocDefs: FROM "allocdefs" USING [
    AddSwapStrategy, CantSwap, SwappingProcedure, SwapStrategy],
  AltoDefs: FROM "altodefs" USING [PageCount, PageSize, wordlength],
  AltoFileDefs: FROM "altofiledefs" USING [KD, SN, vDA],
  BootDefs: FROM "bootdefs" USING [BootFile],
  DirectoryDefs: FROM "directorydefs" USING [DirectoryLookup],
  DiskDefs: FROM "diskdefs" USING [SetDisk],
  DiskKDDefs: FROM "diskkddefs",
  ImageDefs: FROM "imagedefs" USING [
    AddCleanupProcedure, CleanupItem, CleanupMask, CleanupProcedure],
  InlineDefs: FROM "inlinedefs" USING [BITAND, BITNOT, BITOR, BITSHIFT],
  NucleusDefs: FROM "nucleusdefs",
  SegmentDefs: FROM "segmentdefs" USING [
    DefaultBase, FileNameError, FileSegmentAddress, FileSegmentHandle,
    MoveFileSegment, NewFileSegment, Read, SwapIn, SwapOut, SwapUp, Unlock,
    Write];

DEFINITIONS FROM AltoDefs, AltoFileDefs, SegmentDefs;

DiskKD: PROGRAM
  IMPORTS AllocDefs, BootDefs, DirectoryDefs, DiskDefs, ImageDefs, SegmentDefs
  EXPORTS DiskKDDefs, NucleusDefs =

  BEGIN

  InitializeDiskKD: PUBLIC PROCEDURE =
    BEGIN
    nameKD: STRING = "DiskDescriptor."L;
    pages: PageCount;
    IF ~DirectoryDefs.DirectoryLookup[@diskKD.file.fp, nameKD, FALSE]
      THEN SIGNAL FileNameError[nameKD];
    MoveFileSegment[diskKD, DefaultBase, 1];
    OpenDiskKD[];
    DiskDefs.SetDisk[@kd.disk];
    pages ← (kd.size+PageSize-1)/PageSize;
    [] ← CloseDiskKD[];
    MoveFileSegment[diskKD, DefaultBase, pages];
    RETURN
    END;

  OpenDiskKD: PROCEDURE =
    BEGIN
    IF ~diskKD.swappedin THEN
      BEGIN  SwapIn[diskKD];
      kd ← FileSegmentAddress[diskKD];
      kd.changed ← 0;
      END
    ELSE swapKD.proc ← AllocDefs.CantSwap;
    RETURN
    END;

  UpdateDiskKD: PUBLIC PROCEDURE =
    BEGIN
    IF diskKD.swappedin
    AND kd.changed#0 THEN
      BEGIN
      diskKD.write ← TRUE;
      SwapUp[diskKD];
      diskKD.write ← FALSE;
      kd.changed ← 0;
      END;
    RETURN
    END;

  CloseDiskKD: PUBLIC PROCEDURE RETURNS [BOOLEAN] =
    BEGIN
    IF ~diskKD.swappedin THEN RETURN[FALSE];
    swapKD.proc ← AllocDefs.CantSwap;  UpdateDiskKD[];
    Unlock[diskKD];  SwapOut[diskKD];
    RETURN[TRUE]
    END;

  DiskKDSwapper: AllocDefs.SwappingProcedure =
```

```
    BEGIN
    RETURN[CloseDiskKD[]];
    END;

  CleanupDiskKD: PUBLIC ImageDefs.CleanupProcedure =
    BEGIN
    SELECT why FROM
      Finish, Abort, OutLd => [] ← CloseDiskKD[];
      -- Save =>
      --   We depend on MakeImage to call CloseDiskKD when
      --   it has finished allocating the image file pages.
      --   Logically, it should also call ResetDisk at this
      --   time, but it can't do that until the Restore.
      -- Restore =>
      --   We depend on MakeImage to call InitializeDiskKD
      --   as soon as the image file starts up so that the
      --   Real to Virtual disk address map can be set up.
      ENDCASE;
    RETURN
    END;


  AllOnes: WORD = 177777B;

  NewSN: PUBLIC PROCEDURE RETURNS [sn:SN] =
    BEGIN  OpenDiskKD[];
    IF (kd.lastSN.part2 ← kd.lastSN.part2+1) = 0
      THEN kd.lastSN.part1 ← kd.lastSN.part1+1;
    sn ← kd.lastSN;  kd.changed ← AllOnes;
    swapKD.proc ← DiskKDSwapper;
    RETURN
    END;

  BitAddress: TYPE = RECORD [word:[0..7777B], bit:[0..17B]];

  DiskFull: PUBLIC SIGNAL = CODE;

  AssignDiskPage: PUBLIC PROCEDURE [da:vDA]
    RETURNS [vDA] =
    BEGIN OPEN InlineDefs;
    onebit: WORD;
    wa: CARDINAL;
    ba: [0..16);
    w: POINTER TO WORD;
    base: BitAddress = LOOPHOLE[da+1];
    baseWa: CARDINAL ← base.word;
    baseBa: CARDINAL ← base.bit;
    OpenDiskKD[];
    DO ENABLE UNWIND => swapKD.proc ← DiskKDSwapper;
      FOR wa IN [baseWa..kd.size) DO
        IF (w ← @kd.table[wa])↑ # AllOnes THEN
          FOR ba IN [baseBa..wordlength) DO
            onebit ← BITSHIFT[100000B,-ba];
            IF BITAND[w↑,onebit]=0 THEN
              BEGIN
              w↑ ← BITOR[w↑,onebit];
              kd.changed ← AllOnes;
              kd.freePages ← MAX[kd.freePages,1]-1;
              swapKD.proc ← DiskKDSwapper;
              RETURN[vDA[wa*wordlength+ba]];
              END;
            ENDLOOP;
        baseBa ← 0;
        ENDLOOP;
      IF baseWa=0 THEN
        BEGIN
        [] ← CloseDiskKD[];
        SIGNAL DiskFull;
        OpenDiskKD[];
        END;
      baseWa ← 0;
      ENDLOOP;
    END;

  ReleaseDiskPage: PUBLIC PROCEDURE [v:vDA] =
    BEGIN OPEN InlineDefs;
```

```
    word: POINTER TO WORD;
    OpenDiskKD[];
    word ← @kd.table[LOOPHOLE[v,BitAddress].word];
    word↑ ← BITAND[word↑,
      BITNOT[BITSHIFT[100000B,-LOOPHOLE[v,BitAddress].bit]]];
    kd.changed ← AllOnes;
    kd.freePages ← kd.freePages+1;
    swapKD.proc ← DiskKDSwapper;
    RETURN
    END;

  CountFreeDiskPages: PUBLIC PROCEDURE RETURNS [count: CARDINAL] =
    BEGIN
    OpenDiskKD[];
    count ← kd.freePages;
    swapKD.proc ← DiskKDSwapper;
    RETURN
    END;

  kd: POINTER TO KD;
  swapKD: AllocDefs.SwapStrategy;
  diskKD: FileSegmentHandle;
  cleanupKD: ImageDefs.CleanupItem;

  swapKD.proc ← AllocDefs.CantSwap;
  cleanupKD.proc ← CleanupDiskKD;
  cleanupKD.mask ← ImageDefs.CleanupMask[Finish] +
    ImageDefs.CleanupMask[Abort] + ImageDefs.CleanupMask[OutLd];
  diskKD ← NewFileSegment[BootDefs.BootFile[Read+Write],DefaultBase,1,Read];
  AllocDefs.AddSwapStrategy[@swapKD];
  ImageDefs.AddCleanupProcedure[@cleanupKD];
  InitializeDiskKD[];

  -- Should we support running without a DiskDescriptor?

  END.
```