```
-- DiskIO.Mesa  Edited by Sandman on May 12, 1978  2:11 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [BYTE, PageNumber, PageSize],
  AltoFileDefs: FROM "altofiledefs" USING [
    DISK, eofDA, fillinDA, SN, vDA, vDC],
  DiskDefs: FROM "diskdefs" USING [
    CB, CBinit, CBptr, CBZ, CBZptr, DA, DC, DDC, DiskPageDesc, DiskRequest,
    DL, DS, DSfakeStatus, DSfreeStatus, DSgoodStatus, DSmaskStatus, FID,
    InvalidDA, lCBZ, nCB, nDisks, nHeads, nSectors, nTracks, RetryCount],
  InlineDefs: FROM "inlinedefs" USING [BITAND, COPY, DIVMOD],
  MiscDefs: FROM "miscdefs",
  NucleusDefs: FROM "nucleusdefs",
  ProcessDefs: FROM "processdefs" USING [DisableInterrupts, EnableInterrupts];

DEFINITIONS FROM DiskDefs;

DiskIO: PROGRAM EXPORTS DiskDefs, MiscDefs, NucleusDefs SHARES DiskDefs = BEGIN

  PageNumber: TYPE = AltoDefs.PageNumber;
  DISK: TYPE = AltoFileDefs.DISK;
  SN: TYPE = AltoFileDefs.SN;
  vDA: TYPE = AltoFileDefs.vDA;
  vDC: TYPE = AltoFileDefs.vDC;
  DC: TYPE = DiskDefs.DC;
  DS: TYPE = DiskDefs.DS;
  CBptr: TYPE = DiskDefs.CBptr;
  CBZptr: TYPE = DiskDefs.CBZptr;

  nil: POINTER = LOOPHOLE[0];
  driveNumber: PUBLIC [0..1] ← 0;
  sysdisk: DISK ← DISK[nDisks,nTracks,nHeads,nSectors];
  disk: POINTER TO DISK = @sysdisk;

  Zero: PUBLIC PROCEDURE [p:POINTER, l:CARDINAL] =
    BEGIN
    IF l=0 THEN RETURN;  p↑ ← 0;
    InlineDefs.COPY [from:p, to:p+1, nwords:l-1];
    RETURN
    END;

  SetDisk: PUBLIC PROCEDURE [d:POINTER TO DISK] =
    BEGIN disk↑ ← d↑; RETURN END;

  GetDisk: PUBLIC PROCEDURE RETURNS [POINTER TO DISK] =
    BEGIN RETURN[disk] END;

  ResetDisk: PUBLIC PROCEDURE RETURNS [POINTER TO DISK] =
    BEGIN
    disk↑ ← DISK[nDisks,nTracks,nHeads,nSectors];
    RETURN[disk]
    END;

  VirtualDA: PUBLIC PROCEDURE [da:DA] RETURNS [vDA] =
    BEGIN
    RETURN[IF da = DA[0,0,0,0,0] THEN AltoFileDefs.eofDA ELSE vDA [
      ((da.disk*disk.tracks+da.track)*disk.heads+
       da.head)*disk.sectors+da.sector]];
    END;

  RealDA: PUBLIC PROCEDURE [v:vDA] RETURNS [da:DA] =
    BEGIN
    i: CARDINAL ← v;
    da ← DA[0,0,0,0,0];
    IF v # AltoFileDefs.eofDA THEN
      BEGIN
      [i,da.sector] ← InlineDefs.DIVMOD[i,disk.sectors];
      [i,da.head] ← InlineDefs.DIVMOD[i,disk.heads];
      [i,da.track] ← InlineDefs.DIVMOD[i,disk.tracks];
      [i,da.disk] ← InlineDefs.DIVMOD[i,disk.disks];
      IF i # 0 THEN da ← InvalidDA;
      END;
    RETURN
    END;
```

```
-- Disk transfer "process"

DCseal: AltoDefs.BYTE = 110B;

DCs: ARRAY vDC OF DC = [
  DC[DCseal,DiskRead, DiskRead, DiskRead, 0,0],  -- ReadHLD
  DC[DCseal,DiskCheck,DiskRead, DiskRead, 0,0],  -- ReadLD
  DC[DCseal,DiskCheck,DiskCheck,DiskRead, 0,0],  -- ReadD
  DC[DCseal,DiskWrite,DiskWrite,DiskWrite,0,0],  -- WriteHLD
  DC[DCseal,DiskCheck,DiskWrite,DiskWrite,0,0],  -- WriteLD
  DC[DCseal,DiskCheck,DiskCheck,DiskWrite,0,0],  -- WriteD
  DC[DCseal,DiskCheck,DiskCheck,DiskCheck,1,0],  -- SeekOnly
  DC[DCseal,DiskCheck,DiskCheck,DiskCheck,0,0]]; -- DoNothing

nextDiskCommand: POINTER TO CBptr = LOOPHOLE[521B];
diskStatus: POINTER TO DS = LOOPHOLE[522B];
lastDiskAddress: POINTER TO DA = LOOPHOLE[523B];
sectorInterrupts: POINTER TO CARDINAL = LOOPHOLE[524B];

-- DoDiskCommand assumes that the version number in a FID (and
-- in an FP) will never be used (is always one).  It further
-- assumes that if fp is nil (zero), a FreePageFID was meant;
-- this allows the rest of the world to use short (3 word) FPs.

FreePageFID: FID = FID[-1,SN[1,1,1,17777B,-1]];

NonZeroWaitCell: WORD ← 1;
waitCell: POINTER TO WORD ← @NonZeroWaitCell;

ResetWaitCell: PUBLIC PROCEDURE =
  BEGIN
  waitCell ← @NonZeroWaitCell;
  END;

SetWaitCell: PUBLIC PROCEDURE [p: POINTER TO WORD] RETURNS [preval: POINTER TO WORD] =
  BEGIN
  ProcessDefs.DisableInterrupts[];
  preval ← waitCell;
  waitCell ← p;
  ProcessDefs.EnableInterrupts[];
  RETURN;
  END;

DoDiskCommand: PUBLIC PROCEDURE [arg:POINTER TO DiskDefs.DDC] =
  BEGIN OPEN arg;
  ptr, next, prev: CBptr;
  la: POINTER TO DL;
  zone: CBZptr = cb.zone;
  cb.headerAddress ← @cb.header;
  IF (la ← cb.labelAddress) = nil THEN
    cb.labelAddress ← la ← @cb.label;
  cb.dataAddress ← ca;
  IF cb.normalWakeups = 0 THEN cb.normalWakeups ← zone.normalWakeups;
  IF cb.errorWakeups = 0 THEN cb.errorWakeups ← zone.errorWakeups;
  IF fp = nil THEN la.fileID ← FreePageFID
  ELSE la.fileID ← FID[1,fp.serial];
  la.page ← cb.page ← page;
  IF da # AltoFileDefs.fillinDA THEN cb.header.diskAddress ← RealDA[da];
  IF restore THEN cb.header.diskAddress.restore ← 1;
  cb.command ← DCs[action];
  cb.command.exchange ← driveNumber;
  prev ← PrevCB[zone];
  -- Put the command on the disk controller's queue
  UNTIL waitCell↑ # 0 DO NULL ENDLOOP; -- Wait for Trident to finish
  ProcessDefs.DisableInterrupts[];
  IF (next ← nextDiskCommand↑) # nil THEN
    BEGIN
    DO  ptr ← next;  next ← ptr.nextCB;
      IF next = nil THEN EXIT;
      ENDLOOP;
    ptr.nextCB ← cb;
    END;
  -- Take care of a possible race with disk controller.  The disk
  -- may have gone idle (perhaps due to an error) even as we were
  -- adding a command to the chain.  To make sure there was no
  -- error, we check the status of the previous cb in this zone.
```

```
    IF nextDiskCommand↑ = nil THEN
      SELECT MaskDS[prev.status, DSmaskStatus] FROM
        DSfreeStatus, DSgoodStatus => nextDiskCommand↑ ← cb;
        ENDCASE;
    ProcessDefs.EnableInterrupts[];
    EnqueueCB[zone,cb];
    RETURN
    END;


  -- Disk command block queue

  InitializeCBstorage: PUBLIC PROCEDURE [
    zone:CBZptr, nCBs:CARDINAL, page:PageNumber, init:CBinit] =
    BEGIN
    cb: CBptr;
    i: CARDINAL;
    nq: CARDINAL = nCBs+1;
    length: CARDINAL = SIZE[CBZ]+nCBs*(SIZE[CB]+SIZE[CBptr]);
    queue: DESCRIPTOR FOR ARRAY OF CBptr ←
      DESCRIPTOR[@zone.queueVec,nq];
    cbVector: DESCRIPTOR FOR ARRAY OF CB ←
      DESCRIPTOR[@zone.queueVec+SIZE[CBptr]*nq,nCBs];
    IF init = clear THEN Zero[zone,length];
    zone.currentPage ← page;  zone.cbQueue ← queue;
    zone.qTail ← 0;  zone.qHead ← 1;
    queue[0] ← NIL; -- end of queue;
    FOR i IN [1..nCBs] DO
      queue[i] ← cb ← @cbVector[i-1];
      cb.zone ← zone;  cb.status ← DSfreeStatus;
      ENDLOOP;
    RETURN
    END;

  NumCBs: PROCEDURE [zone:CBZptr] RETURNS [CARDINAL] =
    BEGIN
    RETURN[LENGTH[zone.cbQueue]-1]
    END;

  ClearCB: PROCEDURE [cb:CBptr] =
    BEGIN
    zone: CBZptr = cb.zone;
    Zero[cb,SIZE[CB]];
    cb.zone ← zone;
    RETURN
    END;

  EnqueueCB: PROCEDURE [zone:CBZptr, cb:CBptr] =
    BEGIN  i: CARDINAL ← zone.qTail;
    IF zone.cbQueue[i] # NIL THEN ERROR;
    zone.cbQueue[i] ← cb;
    IF (i ← i+1) = LENGTH[zone.cbQueue] THEN i ← 0;
    zone.qTail ← i;
    RETURN
    END;

  DequeueCB: PROCEDURE [zone:CBZptr] RETURNS [cb:CBptr] =
    BEGIN  i: CARDINAL ← zone.qHead;
    IF (cb ← zone.cbQueue[i]) = NIL THEN ERROR;
    zone.cbQueue[i] ← NIL;
    IF (i ← i+1) = LENGTH[zone.cbQueue] THEN i ← 0;
    zone.qHead ← i;
    RETURN
    END;

  PrevCB: PROCEDURE [zone:CBZptr] RETURNS [cb:CBptr] =
    BEGIN  i: CARDINAL ← zone.qTail;
    i ← (IF i=0 THEN LENGTH[zone.cbQueue] ELSE i) - 1;
    IF (cb ← zone.cbQueue[i]) = NIL THEN ERROR;
    RETURN
    END;

  CleanupCBqueue: PUBLIC PROCEDURE [zone:CBZptr] =
    BEGIN
    cb: CBptr;
    UNTIL zone.cbQueue[zone.qHead] = NIL DO
```

```
    cb ← GetCB[zone,dontClear];
    ENDLOOP;
  RETURN
  END;


-- Removing CBs from the queue.  If for some reason the disk has
-- gone idle without executing the command, we fake an error
-- in it so that the entire zone of CBs will get retryed.

RetryableDiskError: PUBLIC SIGNAL [cb:CBptr] = CODE;
UnrecoverableDiskError: PUBLIC SIGNAL [cb:CBptr] = CODE;

MaskDS: PROCEDURE [DS, DS] RETURNS [DS] = LOOPHOLE[InlineDefs.BITAND];

GetCB: PUBLIC PROCEDURE [zone:CBZptr, init:CBinit] RETURNS [cb:CBptr] =
  BEGIN
  s:DS;  da:vDA;  ddc:DDC;  ec:CARDINAL;
  cb ← DequeueCB[zone];
  UNTIL cb.status.done # 0 DO
    -- not zero means done or fake or free
    IF nextDiskCommand↑ = nil
    AND cb.status.done = 0 THEN
      cb.status ← DSfakeStatus;
    ENDLOOP;
  cb.command.seal ← 0; -- remove command seal
  s ← MaskDS[cb.status, DSmaskStatus];
  SELECT s FROM
    DSgoodStatus =>
      BEGIN
      IF cb.header.diskAddress.restore=0 THEN
        BEGIN
        zone.errorCount ← 0;
        zone.currentBytes ← cb.labelAddress.bytes;
        IF zone.cleanup # LOOPHOLE[0] THEN zone.cleanup[cb];
        END;
      IF init = clear THEN ClearCB[cb];
      END;
    DSfreeStatus =>
      ClearCB[cb]; -- really means DSneverBeenUsed
    ENDCASE =>
      BEGIN -- some error occurred
      -- busy wait until disk controller is idle
      UNTIL nextDiskCommand↑ = nil DO NULL ENDLOOP;
      ec ← zone.errorCount ← zone.errorCount+1;
      IF ec >= RetryCount THEN ERROR UnrecoverableDiskError[cb];
      da ← zone.errorDA ← VirtualDA[cb.header.diskAddress];
      IF cb.status.finalStatus = CheckError THEN zone.checkError ← TRUE;
      InitializeCBstorage [
        zone,NumCBs[zone],cb.page,dontClear];
      IF ec > RetryCount/2 THEN
        BEGIN -- start a restore before signalling the error
        lastDiskAddress↑ ← InvalidDA;
        ddc ← DDC[GetCB[zone,clear],nil,da,0,NIL,TRUE,SeekOnly];
        DoDiskCommand[@ddc];
        END;
      ERROR RetryableDiskError[cb];
      END;
  RETURN
  END;

-- Don't all Cleanup procedures need to be locked?
```

```
  -- Segment swapper

  -- Note that each CB is used twice: first to hold the disk label
  -- for page i-1, and then to hold the DCB for page i.  It isn't
  -- reused until the DCB for page i-1 is correctly done, which
  -- is guaranteed to be after the disk label for page i-1 is no
  -- longer needed, since things are done strictly sequentially by
  -- page number.

  -- Currently, DiskRequest.lastAction is not used by SwapPages.

  DiskCheckError: PUBLIC SIGNAL [page:PageNumber] = CODE;

  SwapPages: PUBLIC PROCEDURE [arg:POINTER TO swap DiskDefs.DiskRequest]
    RETURNS [PageNumber, CARDINAL] =
    BEGIN OPEN arg;
    i: PageNumber;
    cb, nextcb: CBptr;
    cbzone: ARRAY [0..1CBZ) OF UNSPECIFIED;
    zone: CBZptr = @cbzone[0];
    ddc: DiskDefs.DDC ← DiskDefs.DDC[,ca,da↑,,fp,FALSE,action];
    InitializeCBstorage[zone,nCB,firstPage,clear];
    IF desc # NIL THEN
      BEGIN zone.info ← desc;
      zone.cleanup ← GetDiskPageDesc;
      END;
    BEGIN
      ENABLE RetryableDiskError --[cb]-- =>
        BEGIN
        ddc.da ← zone.errorDA;
        ddc.ca ← cb.dataAddress;
        RETRY END;
      cb ← GetCB[zone,clear];
      FOR i ← zone.currentPage, i+1 UNTIL i=lastPage+1 DO
        IF ddc.da = AltoFileDefs.eofDA THEN EXIT;
        IF signalCheckError AND zone.errorCount = RetryCount/2
          THEN SIGNAL DiskCheckError[i];
        nextcb ← GetCB[zone,clear];
        cb.labelAddress ← LOOPHOLE[@nextcb.header.diskAddress];
        ddc.cb ← cb;  ddc.page ← i;
        IF i # zone.currentPage THEN ddc.da ← AltoFileDefs.fillinDA;
        DoDiskCommand[@ddc];
        IF ~fixedCA THEN ddc.ca ← ddc.ca+AltoDefs.PageSize;
        cb ← nextcb;
        ENDLOOP;
      CleanupCBqueue[zone];
      END; -- of enable block
    RETURN[i-1,zone.currentBytes]
    END;

    GetDiskPageDesc: PROCEDURE [cb:CBptr] =
      BEGIN
      la: POINTER TO DL = cb.labelAddress;
      desc: POINTER TO DiskPageDesc ← cb.zone.info;
      desc↑ ← DiskPageDesc [
        VirtualDA[la.prev],
        VirtualDA[cb.header.diskAddress],
        VirtualDA[la.next],
        la.page, la.bytes];
      RETURN
      END;

  END..
```