```
-- BFS.Mesa  Edited by Sandman on August 23, 1977  9:47 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs",
  BFSDefs: FROM "bfsdefs",
  DiskDefs: FROM "diskdefs",
  DiskKDDefs: FROM "diskkddefs",
  InlineDefs: FROM "inlinedefs",
  MiscDefs: FROM "miscdefs",
  SegmentDefs: FROM "segmentdefs",
  StringDefs: FROM "stringdefs";

DEFINITIONS FROM AltoDefs, AltoFileDefs, DiskDefs;

BFS: PROGRAM
  IMPORTS DiskDefs, DiskKDDefs, MiscDefs, SegmentDefs, StringDefs
  EXPORTS BFSDefs = BEGIN

  -- These should be POINTER TO ARRAY OF ...
  CAvec: TYPE = DESCRIPTOR FOR ARRAY OF POINTER;
  DAvec: TYPE = DESCRIPTOR FOR ARRAY OF vDA;

  ActOnPages: PUBLIC PROCEDURE [arg:POINTER TO update DiskRequest]
    RETURNS [page:PageNumber, bytes:CARDINAL] =
    BEGIN OPEN arg, DiskDefs;  a: vDC;  ddc: DDC;
    i: PageNumber;  cb, nextcb: CBptr;
    cbzone: ARRAY [0..1CBZ) OF UNSPECIFIED;
    zone: CBZptr = @cbzone[0];
    CAs: CAvec = DESCRIPTOR[ca,lastPage+1];
    DAs: DAvec = DESCRIPTOR[da,lastPage+2];
    InitializeCBstorage[zone,nCB,firstPage,clear];
    zone.info ← da;  zone.cleanup ← cleanup;
    BEGIN ENABLE RetryableDiskError => RETRY;
      cb ← GetCB[zone,clear ! ANY => ERROR];
      FOR i ← zone.currentPage, i+1 UNTIL i=lastPage+1 DO
        BEGIN  -- inner compound to skip DoNothing pages
          a ← IF i=lastPage THEN lastAction ELSE action;
          IF a = DoNothing THEN GOTO SkipThisPage;
          IF DAs[i] = eofDA THEN EXIT;
          IF signalCheckError AND zone.errorCount = RetryCount/2
            THEN SIGNAL DiskCheckError[i];
          nextcb ← GetCB[zone,clear];
          cb.labelAddress ← IF DAs[i+1] = fillinDA
            THEN LOOPHOLE[@nextcb.header.diskAddress]
            ELSE @nextcb.label;
          ddc ← DDC [
            cb,IF fixedCA THEN ca ELSE CAs[i],DAs[i],i,fp,FALSE,a];
          DoDiskCommand[@ddc];
          cb ← nextcb;
          EXITS
            SkipThisPage => NULL;
          END;
        ENDLOOP;
      CleanupCBqueue[zone];
      END; -- of enable block
    RETURN[i-1,zone.currentBytes]
    END;

  GetNextDA: PUBLIC PROCEDURE [cb:CBptr] =
    BEGIN
    pn: PageNumber = cb.page;
    DAs: DAvec = DESCRIPTOR[cb.zone.info,pn+2];
    IF DAs[pn+1] = fillinDA THEN
      DAs[pn+1] ← VirtualDA[cb.labelAddress.next];
    IF DAs[pn-1] = fillinDA THEN
      DAs[pn-1] ← VirtualDA[cb.labelAddress.prev];
    RETURN
    END;
```

```
-- Currently DiskRequest.action is not used by WritePages (WriteD is assumed).
-- Note also that lastAction is used only if lastPage isn't being rewritten.

WritePages: PUBLIC PROCEDURE [arg:POINTER TO extend DiskRequest]
  RETURNS [page:PageNumber, bytes:CARDINAL] = BEGIN
  aop: update DiskRequest;
  firstNewPage: PageNumber;
  local: extend DiskRequest ← arg↑;
  DAs: DAvec = DESCRIPTOR[arg.da,arg.lastPage+2];
  BEGIN OPEN local;
  IF DAs[firstPage] = fillinDA THEN firstNewPage ← firstPage
  ELSE BEGIN
    aop ← DiskRequest [
      ca, da, firstPage, lastPage, fp, fixedCA, WriteD,
      lastAction, signalCheckError, update[GetNextDA]];
    [page,bytes] ← ActOnPages[@aop];
    IF (firstPage ← page) = lastPage
    AND (lastAction # WriteD
    OR bytes = lastBytes) THEN RETURN;
    firstNewPage ← firstPage+1;
    END;
  IF firstNewPage <= lastPage THEN
    BEGIN  aop.da ← da;
    aop.firstPage ← firstNewPage;
    aop.lastPage ← lastPage;
    AssignPages[@aop];
    END;
  [page,bytes] ← RewritePages[@local];
  RETURN
  END; END;

-- Note that only da, firstPage, and lastPage are valid on entry.

AssignPages: PROCEDURE [arg:POINTER TO update DiskRequest] =
  BEGIN OPEN SegmentDefs, arg;  i: PageNumber;
  DAs: DAvec = DESCRIPTOR[da,lastPage+2];
  sink: DataSegmentHandle = NewDataSegment[DefaultBase,1];
  arg↑ ← DiskRequest [
    DataSegmentAddress[sink],,,,NIL,TRUE,ReadLD,
    ReadLD,FALSE,update[CheckFreePage]];
  UNTIL firstPage > lastPage DO
    ENABLE UNWIND => DeleteDataSegment[sink];
    FOR i IN [firstPage..lastPage] DO
      DAs[i] ← DiskKDDefs.AssignDiskPage[DAs[i-1]];
      ENDLOOP;
    i ← firstPage;
    [] ← ActOnPages[arg ! UnrecoverableDiskError--[cb]-- =>
      BEGIN  -- skip bad spots and press on
      firstPage ← cb.page;
      DAs[firstPage] ← fillinDA;
      firstPage ← firstPage+1;
      RETRY
      END];
    firstPage ← i;
    FOR i IN [firstPage..lastPage] DO
      IF (DAs[firstPage] ← DAs[i]) # fillinDA
        THEN firstPage ← firstPage+1;
      ENDLOOP;
    ENDLOOP;
  DeleteDataSegment[sink];
  RETURN
  END;

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

CheckFreePage: PROCEDURE[cb:CBptr] =
  BEGIN
  DAs: POINTER TO ARRAY [0..1) OF vDA = cb.zone.info;
  IF cb.labelAddress.fileID # FreePageFID
    THEN DAs↑[cb.page] ← fillinDA;
  RETURN
  END;
```

```
  -- Note that action and lastAction are not used (WriteLD is assumed).

  RewritePages: PUBLIC PROCEDURE [arg:POINTER TO extend DiskRequest]
    RETURNS [PageNumber, CARDINAL] =
    BEGIN OPEN arg;  i: PageNumber;
    cbzone: ARRAY [0..1CBZ) OF UNSPECIFIED;
    zone: CBZptr = @cbzone[0];  cb: CBptr;
    CAs: CAvec = DESCRIPTOR[ca,lastPage+1];
    DAs: DAvec = DESCRIPTOR[da,lastPage+2];
    ddc: DDC ← DDC[,ca,,,fp,FALSE,WriteLD];
    InitializeCBstorage[zone,nCB,firstPage,clear];
    BEGIN ENABLE RetryableDiskError => RETRY;
      FOR i ← zone.currentPage, i+1 UNTIL i=lastPage+1 DO
        cb ← GetCB[zone,clear];
        IF (i = lastPage AND lastBytes # CharsPerPage)
        OR DAs[i+1] = fillinDA THEN DAs[i+1] ← eofDA;
        cb.label.next ← RealDA[DAs[i+1]];
        cb.label.prev ← RealDA[DAs[i-1]];
        cb.label.bytes ←
          IF i = lastPage THEN lastBytes ELSE CharsPerPage;
        ddc.cb ← cb;  ddc.da ← DAs[i];  ddc.page ← i;
        IF ~fixedCA THEN ddc.ca ← CAs[i];
        DoDiskCommand[@ddc];
        ENDLOOP;
      CleanupCBqueue[zone];
      END;
    RETURN[lastPage,lastBytes]
    END;

  jump: CARDINAL = 10*nSectors;

  CreatePages: PUBLIC PROCEDURE [
    ca:POINTER, cfa:POINTER TO CFA,
    lastPage:PageNumber, lastBytes:CARDINAL] =
    BEGIN
    da: vDA ← cfa.fa.da;
    arg: extend DiskRequest;
    DAs: ARRAY [-1..jump] OF vDA;
    page: PageNumber ← cfa.fa.page;
    DO -- until lastPage is written
      MiscDefs.SetBlock[@DAs[-1],fillinDA,jump+2];  DAs[0] ← da;
      arg ← DiskRequest [
        ca,@DAs[-page],page,MIN[lastPage,page+(jump-1)],
        @cfa.fp,TRUE,WriteD,WriteD,FALSE,extend[lastBytes]];
      [] ← WritePages[@arg];
      da ← DAs[arg.lastPage-page];
      page ← arg.lastPage;
      IF page = lastPage THEN EXIT;
      ENDLOOP;
    cfa.fa ← FA[da,lastPage,lastBytes];
    RETURN
    END;

  DeletePages: PUBLIC PROCEDURE [
    ca:POINTER, fp:POINTER TO FP, da:vDA, page:PageNumber] =
    BEGIN
    arg: update DiskRequest;
    lastPage, i: PageNumber;
    DAs: ARRAY [-1..jump] OF vDA;
    UNTIL da=eofDA DO
      MiscDefs.SetBlock[@DAs[-1],fillinDA,jump+2];
      DAs[0] ← da;
      arg ← DiskRequest [
        ca,@DAs[-page],page,page+(jump-1),fp,TRUE,
        ReadD,ReadD,FALSE,update[GetNextDA]];
      lastPage ← ActOnPages[@arg].page;
      MiscDefs.Zero[ca,PageSize];
      arg.fp ← LOOPHOLE[0];
      arg.lastPage ← lastPage;
      arg.action ← arg.lastAction ← WriteLD;
      [] ← ActOnPages[@arg];
      FOR i IN [0..lastPage-page] DO
        DiskKDDefs.ReleaseDiskPage[DAs[i]];
        ENDLOOP;
      da ← DAs[lastPage+1-page];
      page ← lastPage+1;
```

```
    ENDLOOP;
  RETURN
  END;

CreateFile: PUBLIC PROCEDURE [name:STRING, fp, dirFP:POINTER TO FP] =
  BEGIN OPEN SegmentDefs;
  DAs: ARRAY[-1..2] OF vDA ← [eofDA,fillinDA,fillinDA,eofDA];
  buf: DataSegmentHandle = NewDataSegment[DefaultBase,1];
  ld: POINTER TO LD = DataSegmentAddress[buf];
  arg: extend DiskRequest ← DiskRequest [
    ld,@DAs[0],0,1,fp,TRUE,WriteD,WriteD,FALSE,extend[0]];
  BEGIN ENABLE UNWIND => DeleteDataSegment[buf];
    MiscDefs.Zero[ld,PageSize];
    ld.created ← MiscDefs.DAYTIME[];
    StringDefs.MesaToBcplString[name,LOOPHOLE[@ld.name]];
    ld.propBegin ← @ld.props[0]-ld;
    ld.propLength ← LENGTH[ld.props];
    IF dirFP # NIL THEN MakeCFP[@ld.dirFP,dirFP];
    fp↑ ← FP[DiskKDDefs.NewSN[],eofDA];
    [] ← WritePages[@arg];
    END;
  fp.leaderDA ← DAs[0];
  DeleteDataSegment[buf];
  RETURN
  END;

MakeFP: PUBLIC PROCEDURE [
  fp:POINTER TO FP, cfp:POINTER TO CFP] =
  BEGIN
  fp↑ ← FP[cfp.serial,cfp.leaderDA];
  RETURN
  END;

MakeCFP: PUBLIC PROCEDURE [
  cfp:POINTER TO CFP, fp:POINTER TO FP] =
  BEGIN
  cfp↑ ← CFP[fp.serial,1,0,fp.leaderDA];
  RETURN
  END;

END.
```