```
-- CoreMap.Mesa
-- Edited by
--            Sandman on April 15, 1978  5:28 PM
--            Barbara on June 22, 1978  2:59 PM

DIRECTORY
  AltoDefs: FROM "altodefs" USING [MaxVMPage, PageNumber, PageSize],
  BcdDefs: FROM "bcddefs" USING [MTHandle, MTIndex, NameRecord, NameString],
  BootDefs: FROM "bootdefs",
  ControlDefs: FROM "controldefs" USING [GFT, GlobalFrameHandle],
  DebugData: FROM "debugdata" USING [debugPilot, ESV],
  DebugMiscDefs: FROM "debugmiscdefs" USING [CommandNotAllowed, ControlDEL],
  DebugUtilityDefs: FROM "debugutilitydefs" USING [
    AREAD, LoadStateInvalid, MREAD, SREAD],
  IODefs: FROM "iodefs" USING [
    CR, NumberFormat, Rubout, SP, WriteChar, WriteLine, WriteNumber,
    WriteOctal, WriteString],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [
    BcdBase, EnumerateModuleTable, ReleaseBcdSeg, SetUpBcd],
  LoadStateDefs: FROM "loadstatedefs" USING [
    BcdAddress, BcdSegFromLoadState, ConfigIndex, EnumerateLoadStateBcds,
    FileSegmentHandle, InitializeRelocation, InputLoadState, ReleaseLoadState,
    ReleaseRelocation, Relocation],
  SegmentDefs: FROM "segmentdefs" USING [
    AddressFromPage, DataSegmentHandle, DataSegmentObject, FileSegmentHandle,
    FileSegmentObject, Object, ObjectHandle, ObjectType, PageCount,
    PageNumber, SegmentObject],
  StreamDefs: FROM "streamdefs" USING [ControlDELtyped],
  StringDefs: FROM "stringdefs" USING [AppendSubString, SubStringDescriptor],
  SystemDefs: FROM "systemdefs" USING [
    AllocateHeapNode, AllocateHeapString, AllocateSegment, FreeHeapNode,
    FreeHeapString, FreeSegment, PruneHeap];

DEFINITIONS FROM AltoDefs, SystemDefs, IODefs, SegmentDefs;

CoreMap: PROGRAM
  IMPORTS DDptr: DebugData, DebugMiscDefs, DebugUtilityDefs, LoadStateDefs,
    IODefs, SystemDefs, SegmentDefs, StringDefs, StreamDefs, LoaderBcdUtilDefs
  EXPORTS DebugMiscDefs
  SHARES SegmentDefs =

BEGIN

  byte: NumberFormat = NumberFormat[8,FALSE,TRUE,3];
  word: NumberFormat = NumberFormat[8,FALSE,TRUE,6];

  CopyRead: PROCEDURE [from, to: POINTER, nwords: CARDINAL] =
    BEGIN
    i: CARDINAL;
    FOR i IN [0..nwords) DO
      (to+i)↑ ← DebugUtilityDefs.SREAD[from+i];
      ENDLOOP;
    RETURN
    END;

  PrintPages: PROCEDURE [
    page:PageNumber, count:PageCount, state:PageState] =
    BEGIN
    WriteNumber[page,byte]; WriteChar[SP];
    WriteNumber[AddressFromPage[page],word];
    WriteString["     "L];
    WriteNumber[count,byte];
    WriteLine[SELECT state FROM
      free => " free"L,
      data => " data"L,
      file => " file"L,
      busy => " busy"L,
      ENDCASE => " ?"L];
    RETURN
    END;

  PrintDataSegment: PROCEDURE [seg:DataSegmentHandle] RETURNS [PageCount] =
    BEGIN
    s: DataSegmentObject;
    CopyRead[to: @s, from: seg, nwords: SIZE[DataSegmentObject]];
    PrintPages[s.VMpage,s.pages,data];
```

```
    RETURN[s.pages]
    END;

  PrintFileSegment: PROCEDURE [seg:FileSegmentHandle] RETURNS [PageCount] =
    BEGIN
    s: FileSegmentObject;
    CopyRead[to: @s, from: seg, nwords: SIZE[FileSegmentObject]];
    WriteNumber[s.VMpage,byte]; WriteChar[SP];
    WriteNumber[AddressFromPage[s.VMpage],word]; WriteChar[SP];
    WriteNumber[s.base,byte];  WriteChar[SP];
    WriteNumber[s.pages,byte]; WriteChar[SP];
    WriteString["SN"L];
    WriteOctal[DebugUtilityDefs.AREAD[@s.file.fp.serial.part2]];
    SELECT s.class FROM
      code => WriteString[" code"L];
      ENDCASE;
    IF s.read OR s.write THEN WriteChar[' ];
    IF s.read THEN WriteChar['R];
    IF s.write THEN WriteChar['W];
    IF s.lock > 0 THEN
      BEGIN
      WriteString[" lock="L];
      WriteOctal[s.lock];
      END;
    SELECT s.class FROM
      code => PrintFileName[seg];
      ENDCASE;
    WriteChar[CR];
    RETURN[s.pages]
    END;

  PrintFileName:  PROCEDURE [seg:  FileSegmentHandle] =
    BEGIN
    node:  POINTER TO NameItem;
    found: BOOLEAN ← FALSE;
    FOR node ← NameList, node.next UNTIL node = NIL DO
      IF node.code = seg THEN
        BEGIN
        WriteChar[SP];
        WriteString[node.module];
        found ← TRUE;
        END;
      ENDLOOP;
    RETURN
    END;

  NameItem:  TYPE = RECORD [
     next:  POINTER TO NameItem,
     code: FileSegmentHandle,
     module:  STRING];

  NameList: POINTER TO NameItem ← NIL;

  FindFrameNames: PROCEDURE =
    BEGIN OPEN LoaderBcdUtilDefs, DebugUtilityDefs, LoadStateDefs;
    bcd: BcdBase;
    proc: PROCEDURE [config: ConfigIndex, bcdAddr: BcdAddress]
    RETURNS [BOOLEAN] =
      BEGIN
      rel: Relocation ← InitializeRelocation[config];
      bcdseg: FileSegmentHandle ← BcdSegFromLoadState[config];
      code: FileSegmentHandle;
      FindNames: PROCEDURE [mth: BcdDefs.MTHandle, mti: BcdDefs.MTIndex]
       RETURNS [BOOLEAN] =
        BEGIN OPEN ControlDefs;
        frame: GlobalFrameHandle ← MREAD[@GFT[rel[mth.gfi]].frame];
        code ← MREAD[@frame.codesegment];
        AddName[code, bcd, mth.name];
        RETURN[FALSE];
        END;
      [] ← EnumerateModuleTable[bcd ← SetUpBcd[bcdseg],FindNames];
      ReleaseRelocation[rel];
      ReleaseBcdSeg[bcdseg];
      RETURN[FALSE];
      END;
```

```
    BEGIN
    [] ← InputLoadState[ ! LoadStateInvalid => GOTO noNames];
    [] ← EnumerateLoadStateBcds[recentfirst,proc];
    ReleaseLoadState[];
    EXITS
      noNames => NULL;
    END;
    END;

  AddName: PROCEDURE [code: FileSegmentHandle, bcd: LoaderBcdUtilDefs.BcdBase,
    name: BcdDefs.NameRecord] =
    BEGIN
    ssb: BcdDefs.NameString = LOOPHOLE[bcd+bcd.ssOffset];
    ss: StringDefs.SubStringDescriptor ←
      [base: @ssb.string, offset: name, length: ssb.size[name]];
    s: STRING ← SystemDefs.AllocateHeapString[ssb.size[name]];
    node: POINTER TO NameItem ← AllocateHeapNode[SIZE[NameItem]];
    StringDefs.AppendSubString[s, @ss];
    node↑ ← [next: NameList, code: code, module: s];
    NameList ← node;
    END;

  FreeNames: PROCEDURE =
    BEGIN
    node:  POINTER TO NameItem;
    FOR node ← NameList, NameList UNTIL node = NIL DO
      NameList ← node.next;
      SystemDefs.FreeHeapString[node.module];
      SystemDefs.FreeHeapNode[node];
      ENDLOOP;
    [] ← SystemDefs.PruneHeap[];
    END;

  PageState: TYPE = {free,data,file,busy};

  PageRecord: TYPE = RECORD [
    state: PageState,
    segment: POINTER];

  PageMap: DESCRIPTOR FOR ARRAY OF PageRecord ← DESCRIPTOR[NIL,MaxVMPage+1];

  StuffDataSegment: PROCEDURE [seg:DataSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    i: PageNumber;
    status: PageRecord = PageRecord[data,seg];
    s: DataSegmentObject;
    CopyRead[to: @s, from: seg, nwords: SIZE[DataSegmentObject]];
    FOR i IN [s.VMpage..s.VMpage+s.pages) DO
      PageMap[i] ← status;
      ENDLOOP;
    RETURN[FALSE]
    END;

  StuffFileSegment: PROCEDURE [seg:FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN
    i: PageNumber;
    status: PageRecord;
    s: FileSegmentObject;
    CopyRead[to: @s, from: seg, nwords: SIZE[FileSegmentObject]];
    IF s.swappedin THEN
      BEGIN
      status ← PageRecord[file,seg];
      FOR i IN [s.VMpage..s.VMpage+s.pages) DO
        PageMap[i] ← status;
        ENDLOOP;
      END;
    RETURN[FALSE]
    END;

--utilities

PageFree: PUBLIC PROCEDURE [page:AltoDefs.PageNumber] RETURNS [BOOLEAN] =
  BEGIN OPEN DebugUtilityDefs;
  table: BootDefs.SystemTableHandle = DDptr.ESV.tables;
  pagemap: POINTER TO BootDefs.PageMap ← SREAD[@table.pagemap];
  RETURN[SREAD[@pagemap[page]] = BootDefs.FreePage]
```

```
  END;

SegmentObject: TYPE = SegmentDefs.SegmentObject;

EnumerateDataSegments: PUBLIC PROCEDURE [
  proc:PROCEDURE [DataSegmentHandle] RETURNS [BOOLEAN]]
  RETURNS [DataSegmentHandle] =
  BEGIN OPEN DebugUtilityDefs, SegmentDefs, BootDefs;
  seg: DataSegmentHandle;
  temp: SegmentObject;
  table: BootDefs.SystemTableHandle = DDptr.ESV.tables;
  pagemap: POINTER TO BootDefs.PageMap ← SREAD[@table.pagemap];
  i: [0..AltoDefs.MaxVMPage] ← 0;
  WHILE i  <= AltoDefs.MaxVMPage DO
    seg ← SREAD[@pagemap[i]];
    IF seg # FreePage AND seg # BusyPage THEN
      BEGIN
      CopyRead[to: @temp, from: seg, nwords: SIZE[SegmentObject]];
      WITH t: temp SELECT FROM
        data =>
          BEGIN
          IF proc[seg] THEN RETURN [seg];
          i ← i + t.pages;
          END;
        file => i ← i + t.pages;
        ENDCASE;
      END
    ELSE i ← i + 1;
    ENDLOOP;
  RETURN[NIL];
  END;

EnumerateFileSegments: PROCEDURE [
  proc:PROCEDURE [FileSegmentHandle] RETURNS [BOOLEAN]]
  RETURNS [FileSegmentHandle] =
  BEGIN
  CheckSegment: PROCEDURE [seg: FileSegmentHandle] RETURNS [BOOLEAN] =
    BEGIN OPEN SegmentDefs;
    temp: SegmentObject;
    CopyRead[from: seg, to: @temp, nwords: SIZE[SegmentObject]];
    RETURN[WITH temp SELECT FROM
      file => proc[seg],
      ENDCASE => FALSE]
    END;
  RETURN[LOOPHOLE[EnumerateObjects[segment, LOOPHOLE[CheckSegment]]]]
  END;

ObjectHandle: TYPE = SegmentDefs.ObjectHandle;

EnumerateObjects: PROCEDURE [type: SegmentDefs.ObjectType,
  proc:PROCEDURE [ObjectHandle] RETURNS [BOOLEAN]]
  RETURNS [object: ObjectHandle] =
  BEGIN OPEN DebugUtilityDefs, SegmentDefs, BootDefs;
  temp: Object;
  i, j: CARDINAL;
  table: TableHandle;
  systemtable: BootDefs.SystemTableHandle = DDptr.ESV.tables;
  FOR table ← SREAD[@systemtable.table], SREAD[@table.link]
  UNTIL table = NIL DO
    j ← i ← SIZE[Table];
    FOR object ← @table.free + i, object + i UNTIL j >= AltoDefs.PageSize DO
      CopyRead[to: @temp, from: object, nwords: SIZE[Object]];
      i ← WITH t: temp SELECT FROM
        segment => SELECT t.type FROM
          data => SIZE[data segment Object],
          ENDCASE => SIZE[file segment Object],
        file => SIZE[file Object],
        free => t.size,
        ENDCASE => SIZE[length Object];
      j ← j + i;
      IF temp.tag = type AND proc[object] THEN RETURN[object];
      ENDLOOP;
    ENDLOOP;
  RETURN[NIL]
  END;
```

```
coremap: PUBLIC PROCEDURE =
  BEGIN
  dseg: DataSegmentHandle;
  fseg: FileSegmentHandle;
  count: PageCount ← 0;
  page, hole: PageNumber ← 0;
  state, next: PageState ← free;
  IF DDptr.debugPilot THEN
    BEGIN
    IODefs.WriteString["Debugging Pilot--"];
    SIGNAL DebugMiscDefs.CommandNotAllowed;
    END;
  PageMap ← DESCRIPTOR [AllocateSegment [
    LENGTH[PageMap]*SIZE[PageRecord]],LENGTH[PageMap]];
  BEGIN ENABLE UNWIND =>
    BEGIN
    FreeSegment[BASE[PageMap]];
    FreeNames[];
    END;
    FOR page IN [0..LENGTH[PageMap]) DO
      PageMap[page] ← PageRecord [
        IF PageFree[page] THEN free ELSE busy, NIL];
      ENDLOOP;
    FindFrameNames[];
    [] ← EnumerateDataSegments[StuffDataSegment];
    [] ← EnumerateFileSegments[StuffFileSegment];
    WriteChar[CR];  page ← 0;
    UNTIL page >= LENGTH[PageMap] DO
      ENABLE Rubout => EXIT;
      next ← PageMap[page].state;
      dseg ← fseg ← PageMap[page].segment;
      IF next # state AND count#0 THEN
        BEGIN
        PrintPages[hole,count,state];
        hole ← hole+count; count ← 0;
        END;
      page ← page + (
        SELECT state ← next FROM
          data => PrintDataSegment[dseg],
          file => PrintFileSegment[fseg],
          ENDCASE => 1);
      SELECT state FROM
        data, file => hole ← page;
        ENDCASE => count ← count+1;
      IF StreamDefs.ControlDELtyped[] THEN SIGNAL DebugMiscDefs.ControlDEL;
      REPEAT
        FINISHED => IF count#0 THEN
          PrintPages[hole,count,state];
      ENDLOOP;
    END;
  FreeSegment[BASE[PageMap]];
  FreeNames[];
  WriteChar[CR];
  RETURN
  END;

END...
```