```
-- SegmentDefs.Mesa  Edited by Sandman on March 21, 1978  5:13 PM

DIRECTORY
  AltoDefs: FROM "altodefs",
  AltoFileDefs: FROM "altofiledefs";

DEFINITIONS FROM AltoDefs, AltoFileDefs;

SegmentDefs: DEFINITIONS = BEGIN

  FileHint: TYPE = AltoFileDefs.FH;
  FileIndex: TYPE = AltoFileDefs.FI;
  PageCount: TYPE = AltoDefs.PageCount;
  PageNumber: TYPE = AltoDefs.PageNumber;


  -- Basic Memory Addressing:

  PagePointer: PROCEDURE [a: POINTER] RETURNS [POINTER];
  PageFromAddress: PROCEDURE [a: POINTER] RETURNS [PageNumber];
  AddressFromPage: PROCEDURE [p: PageNumber] RETURNS [POINTER];

  PageFree: PROCEDURE [page: PageNumber] RETURNS [BOOLEAN];
  PagesFree: PROCEDURE [base: PageNumber, pages: PageCount] RETURNS [BOOLEAN];


  -- Primative System Objects:

  ObjectType: PRIVATE TYPE = {free, segment, file, length};

  ObjectHandle: PRIVATE TYPE = POINTER TO Object;

  Object: TYPE = RECORD [
    busy: PRIVATE BOOLEAN,
    body: SELECT tag: ObjectType FROM
      free => [
        seal: PRIVATE [0..37B],
        size: PRIVATE FrobSize,
        fwdp, backp: PRIVATE FrobLink],
      segment => [
        SELECT type: SegmentType FROM
          data => [
            unused: [0..3777B],
            pages: [1..MaxVMPage+1],
            VMpage: [0..MaxVMPage]],
          file => [
            swappedin: BOOLEAN,
            read, write: BOOLEAN,
            class: FileSegmentClass,
            VMpage: [0..MaxVMPage],
            file: FileHandle,
            base: PageNumber,
            pages: [1..MaxVMPage+1],
            lock: LockCount,
            location: SELECT loc: SegmentLocation FROM
              disk => [
                hint: FileHint],
              remote => [
                proc: RemoteSegProc,
                info: UNSPECIFIED],
              ENDCASE],
          ENDCASE],
      file => [
        open: BOOLEAN,
        length: BOOLEAN,
        lengthvalid: BOOLEAN,
        read, write, append: BOOLEAN,
        lock: LockCount,
        lengthchanged: BOOLEAN,
        unused: [0..177B],
        swapcount: RefCount,
        segcount: SegCount,
        fp: FP],
      length => [
        unused: [0..7B],
        byte: [0..BytesPerPage],
        page: PageNumber,
        file: FileHandle,
        da: vDA],
      ENDCASE];

  RefCount: TYPE = [0..MaxRefs];
  MaxRefs: CARDINAL = 255;

  LockCount: TYPE = [0..MaxLocks];
  MaxLocks: CARDINAL = 127;

  SegCount: TYPE = [0..MaxSegs];
  MaxSegs: CARDINAL = 177777B;

  -- Free Objects

  Frob: PRIVATE TYPE = free Object;
  FrobHandle: PRIVATE TYPE = POINTER TO Frob;
  FrobLink: PRIVATE TYPE = ORDERED POINTER [0..AltoDefs.PageSize) TO Frob;
  FrobSize: PRIVATE TYPE = [0..AltoDefs.PageSize);
  FrobNull: PRIVATE FrobLink = LAST[FrobLink];



  -- S E G M E N T S:

  DefaultPages: PageCount = 0;
  DefaultBase: PageNumber = MaxFilePage;

  SegmentObject: TYPE = segment Object;
  SegmentHandle: TYPE = POINTER TO SegmentObject;

  SegmentType: TYPE = {data, file};
  SegmentLocation: TYPE = {disk, remote};
  RemoteSegProc: TYPE =
    PROCEDURE [seg: FileSegmentHandle, command: RemoteSegCommand];
  RemoteSegCommand: TYPE = UNSPECIFIED;

  VMtoSegment: PROCEDURE [a: POINTER] RETURNS [SegmentHandle];
  SegmentAddress: PROCEDURE [seg: SegmentHandle] RETURNS [POINTER];

  -- Data Segments:

  DataSegmentObject: TYPE = data SegmentObject;
  DataSegmentHandle: TYPE = POINTER TO DataSegmentObject;

  NewDataSegment: PROCEDURE [base: PageNumber, pages: PageCount]
    RETURNS [DataSegmentHandle];
  NewFrameSegment: PROCEDURE [pages: PageCount] RETURNS [DataSegmentHandle];
  DeleteDataSegment: PROCEDURE [seg: DataSegmentHandle];

  VMtoDataSegment: PROCEDURE [a: POINTER] RETURNS [DataSegmentHandle];
  DataSegmentAddress: PROCEDURE [seg: DataSegmentHandle] RETURNS [POINTER];

  EnumerateDataSegments: PROCEDURE [
    proc: PROCEDURE [DataSegmentHandle] RETURNS [BOOLEAN]]
    RETURNS [DataSegmentHandle];


  -- File Segments:

  FileSegmentObject: TYPE = file SegmentObject;
  FileSegmentHandle: TYPE = POINTER TO FileSegmentObject;

  FileSegmentClass: TYPE = {code, other};

  InvalidSegmentSize: SIGNAL [pages: PageCount];

  NewFileSegment: PROCEDURE [
    file: FileHandle, base: PageNumber, pages: PageCount, access: AccessOptions]
    RETURNS [FileSegmentHandle];
  MoveFileSegment: PROCEDURE [
    seg: FileSegmentHandle, base: PageNumber, pages: PageCount];
  MapFileSegment: PROCEDURE [
    seg: FileSegmentHandle, file: FileHandle, base: PageNumber];
  DeleteFileSegment: PROCEDURE [seg: FileSegmentHandle];

  VMtoFileSegment: PROCEDURE [a: POINTER] RETURNS [FileSegmentHandle];
  FileSegmentAddress: PROCEDURE [seg: FileSegmentHandle] RETURNS [POINTER];
  GetFileSegmentDA: PROCEDURE [seg: FileSegmentHandle] RETURNS [vDA];
  SetFileSegmentDA: PROCEDURE [seg: FileSegmentHandle, da: vDA];

  EnumerateFileSegments: PROCEDURE [
    proc: PROCEDURE [FileSegmentHandle] RETURNS [BOOLEAN]]
    RETURNS [FileSegmentHandle];


  -- File Segment Swapping:

  SwapError: SIGNAL [seg: FileSegmentHandle];
  SegmentFault: SIGNAL [seg: FileSegmentHandle, pages: PageCount];
  SwapIn, SwapUp, SwapOut, Unlock: PROCEDURE [seg: FileSegmentHandle];

  -- Initializing File and Data Segments

  CopyDataToFileSegment: PROCEDURE [
    dataseg: DataSegmentHandle, fileseg: FileSegmentHandle];
  CopyFileToDataSegment: PROCEDURE [
    fileseg: FileSegmentHandle, dataseg: DataSegmentHandle];
  ChangeDataToFileSegment: PROCEDURE [
    dataseg: DataSegmentHandle, fileseg: FileSegmentHandle];

  InsufficientVM: SIGNAL [needed: PageCount];
  VMnotFree: SIGNAL [base: PageNumber, pages: PageCount];


  -- F I L E S:

  FileObject: TYPE = file Object;
  FileHandle: TYPE = POINTER TO FileObject;
  LengthObject: TYPE = length Object;
  LengthHandle: TYPE = POINTER TO LengthObject;

  AccessOptions: TYPE = [0..7];
    Read: AccessOptions = 1;
    Write: AccessOptions = 2;
    Append: AccessOptions = 4;
-- Delete: AccessOptions = 8;

  VersionOptions: TYPE = [0..3];
    NewFileOnly: VersionOptions = 1;
    OldFileOnly: VersionOptions = 2;

  DefaultAccess: AccessOptions = 0;
  DefaultVersion: VersionOptions = 0;

  FileNameError: SIGNAL [name: STRING];
  InvalidFP: SIGNAL [fp: POINTER TO FP];
  FileError, FileAccessError: SIGNAL [file: FileHandle];

  NewFile: PROCEDURE [
    name: STRING, access: AccessOptions, version: VersionOptions]
    RETURNS [FileHandle];
  InsertFile: PROCEDURE [fp: POINTER TO FP, access: AccessOptions]
    RETURNS [FileHandle];
  OpenFile, CloseFile: PROCEDURE [file: FileHandle];
  LockFile, UnlockFile: PROCEDURE [file: FileHandle];
  ReleaseFile, DestroyFile: PROCEDURE [file: FileHandle];

  GetFileAccess: PROCEDURE [file: FileHandle] RETURNS [access: AccessOptions];
  SetFileAccess: PROCEDURE [file: FileHandle, access: AccessOptions];

  InsertFileLength: PROCEDURE [file: FileHandle, fa: POINTER TO FA];
  GetEndOfFile: PROCEDURE [file: FileHandle]
    RETURNS [page: PageNumber, byte: CARDINAL];
  SetEndOfFile: PROCEDURE [
    file: FileHandle, page: PageNumber, byte: CARDINAL];
  SetFileLength: PROCEDURE [file: FileHandle, fa: POINTER TO FA];
  UpdateFileLength: PROCEDURE [file: FileHandle, fa: POINTER TO FA];
  GetFileLength: PROCEDURE [file: FileHandle, fa: POINTER TO FA];
```

```
JumpToPage: PROCEDURE [
  cfa: POINTER TO CFA, page: PageNumber, buf: POINTER]
  RETURNS [prev,next: vDA];

GetFileFP: PROCEDURE [file: FileHandle, fp: POINTER TO FP];
FindFile: PROCEDURE [fp: POINTER TO FP] RETURNS [FileHandle];

EnumerateFiles: PROCEDURE [
  proc: PROCEDURE [FileHandle] RETURNS [BOOLEAN]]
  RETURNS [file: FileHandle];

END.
```