```
-- FileCache.Mesa
-- Edited by:
--             Sandman on December 8, 1977  10:22 AM
--             Barbara on May 15, 1978  10:54 AM
--             Barbara on August 30, 1978  10:49 AM

DIRECTORY
  AltoFileDefs: FROM "altofiledefs" USING [eofDA, FP, SN],
  BcdDefs: FROM "bcddefs" USING [BCD, MTIndex, VersionID],
  DebugUsefulDefs: FROM "debugusefuldefs",
  DebugUtilityDefs: FROM "debugutilitydefs",
  DirectoryDefs: FROM "directorydefs" USING [EnumerateDirectory],
  InlineDefs: FROM "inlinedefs" USING [COPY],
  SegmentDefs: FROM "segmentdefs" USING [
    AccessOptions, DeleteFileSegment, FileHandle, FileNameError,
    FileSegmentAddress, FileSegmentHandle, InsertFile, InvalidFP, LockFile,
    MoveFileSegment, NewFileSegment, OpenFile, PageCount, PageNumber, Read,
    ReleaseFile, SwapIn, Unlock, UnlockFile],
  StringDefs: FROM "stringdefs" USING [
    AppendString, EquivalentSubStrings, SubStringDescriptor],
  SymbolTableDefs: FROM "symboltabledefs" USING [NoSymbolTable],
  SystemDefs: FROM "systemdefs" USING [AllocateHeapString, FreeHeapString];

DEFINITIONS FROM AltoFileDefs;

FileCache: PROGRAM
IMPORTS DirectoryDefs, SymbolTableDefs, SystemDefs, StringDefs,
  SegmentDefs
EXPORTS DebugUsefulDefs, DebugUtilityDefs
SHARES SegmentDefs =
  BEGIN

  FileHandle: TYPE = SegmentDefs.FileHandle;

  NullFP: FP = FP[SN[0,0,0,0,0],eofDA];

  FCRecord: TYPE = RECORD [
    name: STRING,
    fp: FP];

  FCSize: CARDINAL = 20;
  FCLimit: CARDINAL = FCSize-1;
  FCArray: ARRAY [0..FCSize) OF FCRecord;

  InitFileCache: PROCEDURE =
    BEGIN
    i: CARDINAL;
    FOR i IN [0..FCSize) DO
      FCArray[i] ← FCRecord[NIL,NullFP];
      ENDLOOP;
    cacheInvalid ← FALSE;
    RETURN
    END;

  PromoteFCRecord: PROCEDURE [i: CARDINAL] =
    BEGIN
    ithFCR: FCRecord;
    IF i = FCLimit THEN RETURN;
    ithFCR ← FCArray[i];
    InlineDefs.COPY[to: @FCArray[i], from: @FCArray[i+1], nwords: SIZE[FCRecord]*(FCLimit-i)];
    FCArray[FCLimit] ← ithFCR;
    RETURN
    END;

  CopyFileName: PROCEDURE [name: STRING] RETURNS [copy: STRING] =
    BEGIN
    IF name = NIL THEN RETURN[NIL];
    copy ← SystemDefs.AllocateHeapString[name.length];
    StringDefs.AppendString[copy,name];
    RETURN
    END;

  AddFCRecord: PROCEDURE [name: STRING, fh: FileHandle] =
    BEGIN
    IF FCArray[0].name # NIL THEN
      SystemDefs.FreeHeapString[FCArray[0].name];
    InlineDefs.COPY[to: @FCArray[0], from: @FCArray[1], nwords: SIZE[FCRecord]*(FCSize-1)];
    FCArray[FCLimit].name ← CopyFileName[name];
    IF fh # NIL THEN FCArray[FCLimit].fp ← fh.fp
    ELSE FCArray[FCLimit].fp ← NullFP;
    cacheInvalid ← TRUE;
    RETURN
    END;

  CacheNewFile: PUBLIC PROCEDURE [name: STRING, access: SegmentDefs.AccessOptions] RETURNS [file: FileH
**andle] =
    BEGIN
    i: CARDINAL;
    file ← NIL;
    FOR i DECREASING IN [0..FCSize) DO
      IF FCArray[i].name = NIL THEN
        BEGIN AddFCRecord[name,NIL]; EXIT END;
      IF EquivalentFileNames[name, FCArray[i].name] THEN
        BEGIN PromoteFCRecord[i]; EXIT END;
      REPEAT FINISHED => AddFCRecord[name,NIL];
      ENDLOOP;
    IF FCArray[FCLimit].fp.leaderDA # eofDA AND ~cacheInvalid THEN
      BEGIN OPEN SegmentDefs;
      file ← InsertFile[@FCArray[FCLimit].fp, access];
      OpenFile[file ! InvalidFP => GOTO BadCache];
      RETURN;
      EXITS BadCache =>
        BEGIN
        IF file.segcount = 0 THEN ReleaseFile[file];
        cacheInvalid ← TRUE;
        END;
      END;
    ValidateCache[];
    IF FCArray[FCLimit].fp.leaderDA # eofDA THEN
      file ← SegmentDefs.InsertFile[@FCArray[FCLimit].fp, access]
    ELSE ERROR SegmentDefs.FileNameError[name];
    RETURN
    END;

  FileName: PUBLIC PROCEDURE [name: STRING, file: FileHandle] =
    BEGIN
    localname: STRING ← [40];
    i: CARDINAL;
      BEGIN
      IF cacheInvalid THEN GO TO notincache
      ELSE FOR i DECREASING IN [0..FCSize) DO
        IF FCArray[i].name = NIL THEN GO TO notincache;
        IF FCArray[i].fp = file.fp THEN
          BEGIN
          StringDefs.AppendString[name,FCArray[i].name];
          PromoteFCRecord[i];
          RETURN
          END;
        REPEAT FINISHED => GO TO notincache;
        ENDLOOP;
      EXITS notincache => AddFCRecord[NIL,file];
      END;
    ValidateCache[];
    IF FCArray[FCLimit].name = NIL THEN
      BEGIN
      FOR i DECREASING IN [1..FCSize) DO
        FCArray[i] ← FCArray[i-1];
        ENDLOOP;
      FCArray[0] ← [NIL,NullFP];
      SIGNAL SegmentDefs.InvalidFP[@file.fp]
      END
    ELSE StringDefs.AppendString[name,FCArray[FCLimit].name];
    RETURN
    END;

  EquivalentFileNames: PROCEDURE [n1, n2: STRING] RETURNS [BOOLEAN] =
    BEGIN
    s1,s2: StringDefs.SubStringDescriptor;
    s1 ← [base: n1, offset: 0,
      length: n1.length - (IF n1[n1.length-1] = '. THEN 1 ELSE 0)];
    s2 ← [base: n2, offset: 0,
      length: n2.length - (IF n2[n2.length-1] = '. THEN 1 ELSE 0)];
    RETURN[StringDefs.EquivalentSubStrings[@s1,@s2]]
    END;

  cacheInvalid: BOOLEAN;

  InvalidateFileCache: PUBLIC PROCEDURE =
    BEGIN
    cacheInvalid ← TRUE;
    END;

  ValidateCache: PROCEDURE =
    BEGIN
    i: CARDINAL;
    CheckEntry: PROCEDURE [fp: POINTER TO FP, dirname: STRING] RETURNS[BOOLEAN] =
      BEGIN
      fcr: POINTER TO FCRecord ← @FCArray[FCLimit];
      THROUGH [0..FCSize) DO
        IF fcr.name = NIL THEN
          BEGIN
          IF fcr.fp.leaderDA = eofDA THEN EXIT;
          IF fcr.fp = fp↑ THEN
            fcr.name ← CopyFileName[dirname];
          END
        ELSE
          BEGIN
          IF EquivalentFileNames[fcr.name, dirname] THEN
            fcr.fp ← fp↑;
          END;
        fcr ← LOOPHOLE[fcr - SIZE[FCRecord]];
        ENDLOOP;
      RETURN[FALSE];
      END;

    IF ~cacheInvalid THEN RETURN;
    FOR i IN [0..FCSize) DO
      IF FCArray[i].name # NIL THEN FCArray[i].fp ← NullFP;
      ENDLOOP;
    DirectoryDefs.EnumerateDirectory[CheckEntry];
    cacheInvalid ← FALSE;
    END;

  FindSymbolTable: PUBLIC PROCEDURE [name: STRING]
    RETURNS [file: SegmentDefs.FileHandle, base: SegmentDefs.PageNumber, pages: SegmentDefs.PageCount]
**=
    BEGIN OPEN SymbolTableDefs, SegmentDefs, BcdDefs;
    headerseg: FileSegmentHandle;
    bHeader: POINTER TO BCD;
    mtb, sgb: CARDINAL;
    mti: MTIndex = LOOPHOLE[0];
    file ← CacheNewFile[name, Read
      !FileNameError => ERROR NoSymbolTable[NIL]];
    headerseg ← NewFileSegment[file,1,1,Read];
    SwapIn[headerseg];
    bHeader ← FileSegmentAddress[headerseg];
    BEGIN OPEN bHeader;
      ENABLE UNWIND =>
        BEGIN
        Unlock[headerseg];
        DeleteFileSegment[headerseg];
        END;
      IF versionident # BcdDefs.VersionID OR nModules # 1
        THEN ERROR NoSymbolTable[NIL];
      IF (pages←nPages) # 1 THEN
        BEGIN
        Unlock[headerseg];
        MoveFileSegment[headerseg,1,pages];
        SwapIn[headerseg];
        bHeader ← FileSegmentAddress[headerseg];
        END;
      mtb ← LOOPHOLE[bHeader,CARDINAL]+mtOffset;
      sgb ← LOOPHOLE[bHeader,CARDINAL]+sgOffset;
      base ← ((mtb+mti).sseg+sgb).base;
      pages ← ((mtb+mti).sseg+sgb).pages+((mtb+mti).sseg+sgb).extraPages;
      END;
    Unlock[headerseg];
    LockFile[file];
    DeleteFileSegment[headerseg];
    UnlockFile[file];
    RETURN
    END;

-- Main Body
  InitFileCache[];

  END...
```