```
-- UnNewConfig.mesa; edited by Sandman on Aug 31, 1978 5:04 PM

DIRECTORY
  BcdDefs: FROM "bcddefs" USING [MTHandle, MTIndex, UnboundLink],
  ControlDefs: FROM "controldefs" USING [
    ControlLink, FrameCodeBase, Free, GFT, GFTIndex, GFTItem,
    GlobalFrameHandle, NullGlobalFrame, TrapLink],
  FrameDefs: FROM "framedefs" USING [
    EnumerateGlobalFrames, GlobalFrame, LockCode, UnlockCode, UnNew],
  InlineDefs: FROM "inlinedefs" USING [BITAND],
  LoaderBcdUtilDefs: FROM "loaderbcdutildefs" USING [
    BcdBase, EnumerateModuleTable, ReleaseBcdSeg, SetUpBcd],
  LoaderDefs: FROM "loaderdefs",
  LoadStateDefs: FROM "loadstatedefs" USING [
    BcdAddress, BcdSegFromLoadState, ConfigIndex, EnumerateLoadStateBcds,
    InitializeRelocation, InputLoadState, MapRealToConfig, ReleaseLoadState,
    ReleaseRelocation, Relocation, RemoveConfig, SetUnresolvedImports],
  SegmentDefs: FROM "segmentdefs" USING [
    DeleteFileSegment, FileSegmentAddress, FileSegmentHandle, SwapError,
    SwapIn, SwapUp, Unlock],
  SystemDefs: FROM "systemdefs" USING [FreeSegment];

UnNewConfig: PROGRAM
  IMPORTS FrameDefs, LoadStateDefs, LoaderBcdUtilDefs, SegmentDefs, SystemDefs
  EXPORTS LoaderDefs = BEGIN

  ConfigIndex: TYPE = LoadStateDefs.ConfigIndex;
  Relocation: TYPE = LoadStateDefs.Relocation;
  BcdBase: TYPE = LoaderBcdUtilDefs.BcdBase;
  GlobalFrameHandle: TYPE = ControlDefs.GlobalFrameHandle;
  NullGlobalFrame: GlobalFrameHandle = ControlDefs.NullGlobalFrame;

  UnNewConfig: PUBLIC PROCEDURE [frame: GlobalFrameHandle] =
    BEGIN OPEN LoadStateDefs;
    config: ConfigIndex;
    rel: Relocation;
    bcdseg: SegmentDefs.FileSegmentHandle;
    bcd: BcdBase;
    FindOriginal: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
      BEGIN
      IF f # frame THEN
        BEGIN
        IF SameCode[frame, f]  = identical AND ~f.copied THEN
          BEGIN frame ← f; RETURN[TRUE] END;
        END;
      RETURN[FALSE];
      END;
    IF frame.copied THEN [] ← FrameDefs.EnumerateGlobalFrames[FindOriginal];
    [] ← InputLoadState[];
    [config: config] ← MapRealToConfig[frame.gfi];
    bcdseg ← BcdSegFromLoadState[config];
    bcd ← LoaderBcdUtilDefs.SetUpBcd[bcdseg];
    rel ← InitializeRelocation[config];
    UnBindConfig[config, bcd, rel];
    CleanupFrames[rel];
    CleanupGFT[rel];
    RemoveConfig[rel, config];
    ReleaseRelocation[rel];
    ReleaseLoadState[];
    LoaderBcdUtilDefs.ReleaseBcdSeg[bcdseg];
    RETURN
    END;

  CodeMatch: TYPE = {identical, same, different};

  SameCode: PROCEDURE [f1, f2: GlobalFrameHandle] RETURNS [CodeMatch] =
    BEGIN
    o1, o2: CARDINAL;
    fcb: ControlDefs.FrameCodeBase;
    IF f1.codesegment # f2.codesegment THEN RETURN[different];
    IF f1.codesegment.swappedin THEN
      BEGIN
      SegmentDefs.SwapIn[f1.codesegment];
      o1 ← f1.code.codebase - SegmentDefs.FileSegmentAddress[f1.codesegment];
      SegmentDefs.Unlock[f1.codesegment];
      END
```

```
    ELSE
      BEGIN
      fcb ← f1.code;
      fcb.swappedout ← FALSE;
      o1 ← fcb.offset;
      END;
    IF f2.codesegment.swappedin THEN
      BEGIN
      SegmentDefs.SwapIn[f2.codesegment];
      o2 ← f2.code.codebase - SegmentDefs.FileSegmentAddress[f2.codesegment];
      SegmentDefs.Unlock[f2.codesegment];
      END
    ELSE
      BEGIN
      fcb ← f2.code;
      fcb.swappedout ← FALSE;
      o2 ← fcb.offset;
      END;
    RETURN[IF o1 = o2 THEN identical ELSE same];
    END;

  CleanupFrames: PROCEDURE [rel: Relocation] =
    BEGIN
    frame: GlobalFrameHandle;
    i, ep: CARDINAL;
    nlinks: [0..256);
    alloced: BOOLEAN ← FALSE;
    DestroyCopy: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
      BEGIN
      FrameDefs.UnNew[f];
      RETURN[FALSE];
      END;
    FOR i IN [1..LENGTH[rel]) DO
      [frame: frame, epbase: ep] ← ControlDefs.GFT[rel[i]];
      IF frame = NullGlobalFrame OR ep # 0 THEN LOOP;
      IF frame.alloced THEN
        BEGIN
        FrameDefs.LockCode[frame];
        nlinks ← frame.code.prefix.nlinks;
        FrameDefs.UnlockCode[frame];
        END;
      IF ~EnumerateCopies[frame, DestroyCopy].shared THEN
        SegmentDefs.DeleteFileSegment[frame.codesegment !
          SegmentDefs.SwapError => CONTINUE];
      IF frame.alloced THEN
        BEGIN
        Align: PROCEDURE [POINTER, WORD] RETURNS [POINTER] =
          LOOPHOLE[InlineDefs.BITAND];
        IF frame.codelinks THEN ControlDefs.Free[frame]
        ELSE ControlDefs.Free[Align[frame - nlinks, 177774B]];
        alloced ← TRUE;
        END;
      ENDLOOP;
    IF ~alloced THEN SystemDefs.FreeSegment[frame];
    RETURN
    END;

  NextMultipleOfFour: PROCEDURE [n: UNSPECIFIED] RETURNS [UNSPECIFIED] =
    BEGIN
    RETURN[n+InlineDefs.BITAND[-n, 3B]]
    END;

  UnBindConfig: PROCEDURE [config: ConfigIndex, bcd: BcdBase, rel: Relocation] =
    BEGIN OPEN LoadStateDefs;
    bindeeRel: Relocation;
    nowUnresolved: BOOLEAN;
    UnBind: PROCEDURE [c: ConfigIndex, b: BcdAddress] RETURNS [BOOLEAN] =
      BEGIN
      bcdseg: SegmentDefs.FileSegmentHandle;
      bindee: BcdBase;
      IF c = config THEN RETURN[FALSE];
      bcdseg ← BcdSegFromLoadState[c];
      bindee ← LoaderBcdUtilDefs.SetUpBcd[bcdseg];
      IF bindee.nImports # 0 THEN
        BEGIN
        bindeeRel ← InitializeRelocation[c];
```

```
      nowUnresolved ← FALSE;
      [] ← LoaderBcdUtilDefs.EnumerateModuleTable[bindee, AdjustLinks];
      SetUnresolvedImports[c, nowUnresolved];
      ReleaseRelocation[bindeeRel];
      END;
    LoaderBcdUtilDefs.ReleaseBcdSeg[bcdseg];
    RETURN[FALSE]
    END;
  AdjustLinks: PROCEDURE [mth: BcdDefs.MTHandle, mti: BcdDefs.MTIndex]
    RETURNS [BOOLEAN] =
    BEGIN
    frame: GlobalFrameHandle = ControlDefs.GFT[bindeeRel[mth.gfi]].frame;
    nlinks: CARDINAL = mth.frame.length;
    changed: BOOLEAN;
    AdjustCopies: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
      BEGIN
      [] ← AdjustFrame[f, mth.code.offset, nlinks];
      RETURN[FALSE];
      END;
    IF frame = NullGlobalFrame OR nlinks = 0 THEN RETURN[FALSE];
    changed ← AdjustFrame[frame, mth.code.offset, nlinks];
    IF changed AND frame.shared AND ~frame.codelinks THEN
      [] ← EnumerateCopies[frame, AdjustCopies];
    IF changed THEN nowUnresolved ← TRUE;
    RETURN[FALSE]
    END;
  AdjustFrame: PROCEDURE [frame: GlobalFrameHandle, offset, nlinks: CARDINAL]
    RETURNS [changed: BOOLEAN] =
    BEGIN
    linkbase: POINTER TO ControlDefs.ControlLink;
    i: CARDINAL;
    changed ← FALSE;
    IF frame.codelinks THEN
      BEGIN OPEN SegmentDefs;
      SwapIn[frame.codesegment];
      linkbase ← FileSegmentAddress[frame.codesegment] + offset;
      END
    ELSE linkbase ← LOOPHOLE[frame];
    FOR i IN [1..nlinks] DO
      BEGIN OPEN ControlDefs;
      link: ControlLink ← (linkbase-i)↑;
      IF BoundHere[link] THEN
        BEGIN
        (linkbase-i)↑ ← (SELECT link.tag FROM
          frame => TrapLink, ENDCASE => BcdDefs.UnboundLink);
        changed ← TRUE;
        END;
      END;
      ENDLOOP;
    IF frame.codelinks THEN
      BEGIN OPEN SegmentDefs;
      seg: FileSegmentHandle = frame.codesegment;
      IF changed THEN
        BEGIN seg.write ← TRUE; SwapUp[seg]; seg.write ← FALSE; END;
      Unlock[seg];
      END;
    RETURN
    END;
  BoundHere: PROCEDURE [link: ControlDefs.ControlLink] RETURNS [BOOLEAN] =
    BEGIN
    i: CARDINAL;
    gfi: ControlDefs.GFTIndex;
    SELECT link.tag FROM
      unbound => RETURN[FALSE];
      ENDCASE => gfi ← FrameDefs.GlobalFrame[link].gfi;
    FOR i IN [0..LENGTH[rel]) DO
      IF rel[i] = gfi THEN RETURN[TRUE];
      ENDLOOP;
    RETURN[FALSE];
    END;

  IF bcd.nExports = 0 AND bcd.nModules # 1 THEN RETURN;
  [] ← EnumerateLoadStateBcds[recentfirst, UnBind];
  RETURN
  END;
```

```
  CleanupGFT: PROCEDURE [rel: Relocation] =
    BEGIN OPEN ControlDefs;
    i: CARDINAL;
    FOR i IN [1..LENGTH[rel]) DO
      GFT[rel[i]] ← GFTItem[NullGlobalFrame, 0];
      ENDLOOP;
    END;

  EnumerateCopies: PROCEDURE [frame: GlobalFrameHandle,
    proc: PROCEDURE [GlobalFrameHandle] RETURNS [BOOLEAN]]
    RETURNS [result: GlobalFrameHandle, shared: BOOLEAN] =
    BEGIN
    FindCopies: PROCEDURE [f: GlobalFrameHandle] RETURNS [BOOLEAN] =
      BEGIN
      IF f = frame THEN RETURN[FALSE];
      SELECT SameCode[frame, f] FROM
        identical =>
          BEGIN
          shared ← TRUE;
          IF f.copied AND proc[f] THEN RETURN[TRUE];
          END;
        same => shared ← TRUE;
        ENDCASE;
      RETURN[FALSE];
      END;
    shared ← FALSE;
    RETURN[FrameDefs.EnumerateGlobalFrames[FindCopies], shared];
    END;

END...
```