```
-- BcdControl.Mesa  Edited by Johnsson on August 30, 1978  9:17 PM

DIRECTORY
  BcdControlDefs: FROM "bcdcontroldefs",
  BcdErrorDefs: FROM "bcderrordefs",
  BcdFileDefs: FROM "bcdfiledefs",
  BcdHeapDefs: FROM "bcdheapdefs",
  BcdLALRDefs: FROM "bcdlalrdefs",
  BcdDefs: FROM "bcddefs",
  BcdTabDefs: FROM "bcdtabdefs",
  BcdTreeDefs: FROM "bcdtreedefs",
  BcdUtilDefs: FROM "bcdutildefs",

  AltoDefs: FROM "altodefs",
  ControlDefs: FROM "controldefs",
  FrameDefs: FROM "framedefs",
  ImageDefs: FROM "imagedefs",
  IODefs: FROM "iodefs",
  MiscDefs: FROM "miscdefs",
  Mopcodes: FROM "mopcodes",
  OsStaticDefs: FROM "osstaticdefs",
  SegmentDefs: FROM "segmentdefs",
  StreamDefs: FROM "streamdefs",
  StringDefs: FROM "stringdefs",
  SymbolCompressorDefs: FROM "symbolcompressordefs",
  TableDefs: FROM "tabledefs",
  TimeDefs: FROM "timedefs",
  TrapDefs: FROM "trapdefs";

DEFINITIONS FROM StreamDefs;

BcdControl: PROGRAM [
  data: BcdControlDefs.BinderData,
  parseg: SegmentDefs.FileSegmentHandle]
  IMPORTS BcdControlDefs, BcdErrorDefs, BcdFileDefs, BcdHeapDefs, BcdLALRDefs, BcdTabDefs, BcdTreeDefs,
** BcdUtilDefs, FrameDefs, ImageDefs, IODefs, MiscDefs, SegmentDefs, StreamDefs, StringDefs, SymbolComp
**ressorDefs, TableDefs, TimeDefs, TrapDefs
  EXPORTS BcdControlDefs
  SHARES OsStaticDefs =
  BEGIN



  -- stream utilities

  FileTooLong: PUBLIC ERROR = CODE;

  shortStreamIndex: PUBLIC PROCEDURE [i: StreamIndex] RETURNS [CARDINAL] =
    BEGIN OPEN AltoDefs;
    -- assumes a normalized index
    IF i.page > VMLimit/CharsPerPage THEN ERROR FileTooLong;
    RETURN [i.page*CharsPerPage + i.byte]
    END;

  longStreamIndex: PUBLIC PROCEDURE [i: CARDINAL] RETURNS [StreamIndex] =
    BEGIN
    RETURN [NormalizeIndex[StreamIndex[page:0, byte:i]]]
    END;

  -- table storage management

  PageCount: TYPE = SegmentDefs.PageCount;
  PageNumber: TYPE = SegmentDefs.PageNumber;

  TableBase: PageNumber = 1;
  TablePageStart: PageCount = 8;
  TablePageStep:  PageCount = 4;
  TablePageLimit: PageCount = 64;

  tableSegment: SegmentDefs.FileSegmentHandle;
  tablePages: PageCount;

  tableOrigin: CARDINAL;
  tableSize: CARDINAL;

  LoadTable: PROCEDURE [nPages: PageCount] =
```

```
    BEGIN OPEN SegmentDefs;
    IF nPages # tablePages THEN
      BEGIN
      IF tablePages # 0 THEN
        BEGIN
        Unlock[tableSegment];
        SwapOut[tableSegment];
        END;
      MoveFileSegment[tableSegment, tableSegment.base, nPages];
      tablePages ← nPages;
      SwapIn[tableSegment ! SegmentFault =>
        BEGIN
        SetEndOfFile[tableSegment.file, tableSegment.base + nPages, 0];
        RETRY
        END];
      END;
    tableOrigin ← LOOPHOLE[FileSegmentAddress[tableSegment]];
    tableSize ← tablePages*AltoDefs.PageSize;
    RETURN
    END;

  InitTableSegment: PROCEDURE =
    BEGIN OPEN SegmentDefs;
    tablefile: FileHandle;
    IF (tablefile←tableRequest.file) = NIL THEN
      tablefile ← NewFile[tableRequest.name, Read+Write+Append, DefaultVersion];
    tableSegment ← NewFileSegment[
      file: tablefile,
      base: TableBase,
      pages: 1,        -- but never swapped as such
      access: Read+Write];
    tablePages ← 0;
    END;

  Initialize: PROCEDURE =
    BEGIN
    RepeatCommand[];
    data.errors ← data.warnings ← aborted ← FALSE;
    data.nModules ← 0;
    data.textIndex ← BcdControlDefs.NullSourceIndex;
    data.currentname ← BcdDefs.NullName;
    -- DisplayOff[black];
    LoadTable[TablePageStart];
    TableDefs.InitializeTable[
      [origin: tableOrigin, size: tableSize], BcdDefs.BinderNTables];
    BcdTabDefs.BcdTabInit[];  BcdTreeDefs.treeinit[];
    BcdHeapDefs.InitHeap[];
    BcdUtilDefs.InitUtilities[];
    RETURN
    END;

  Finalize: PROCEDURE =
    BEGIN
    BcdTreeDefs.treeerase[];
    BcdTabDefs.BcdTabErase[];
    BcdHeapDefs.EraseHeap[];
    BcdUtilDefs.EraseUtilities[];
    TableDefs.EraseTable[];
    -- DisplayOn[];
    data.sourcestream.destroy[data.sourcestream];
    IF comcm # NIL AND (data.errors OR data.warnings) THEN mustwait ← TRUE;
    RETURN
    END;

  RemoveCode: PROCEDURE [link: UNSPECIFIED] =
    BEGIN OPEN FrameDefs;
    SwapOutCode[GlobalFrame[link]];
    RETURN
    END;

  DefaultFileName: PROCEDURE [name, defext: STRING] =
    BEGIN
    i: CARDINAL;
    FOR i IN [0..name.length) DO
      IF name[i] = '. THEN RETURN;
      ENDLOOP;
```

```
    StringDefs.AppendString[name,defext];
    RETURN
    END;

  -- Command gathering

  Cleanup: ImageDefs.CleanupItem ← [link:,proc:RequestFiles,
    mask:ImageDefs.CleanupMask[Save]];
  comcmRequest: short ImageDefs.FileRequest ← ImageDefs.FileRequest[
    link:, file: NIL,
    access: SegmentDefs.Read,
    body: short[,"Com.Cm."]];
  tableRequest: short ImageDefs.FileRequest ← ImageDefs.FileRequest[
    link:, file: NIL,
    access: SegmentDefs.Read+SegmentDefs.Write+SegmentDefs.Append,
    body: short[,"Swatee."]];
  comcm: StreamDefs.StreamHandle ← NIL;
  localPause: BOOLEAN;
  globalPause: BOOLEAN ← TRUE;

  ReadCmdString: PROCEDURE [input, name, switches: STRING] RETURNS [BOOLEAN] =
    BEGIN
    i: CARDINAL ← 0;
    j: CARDINAL;
    activestring: STRING ← name;
    c: CHARACTER;
    name.length ← 0; switches.length ← 0;
    --skip leading blanks --
    WHILE i < input.length DO
      c←input[i];
      IF c # IODefs.SP THEN EXIT;
      i ← i+1;
      ENDLOOP;
    IF i = input.length THEN BEGIN input.length ← 0; RETURN[FALSE] END;
    --parse command--
    FOR i IN [i..input.length) DO
      c←input[i];
      SELECT c FROM
        '/ => activestring ← switches;
        IODefs.SP,IODefs.CR => EXIT;
        ENDCASE => StringDefs.AppendChar[activestring,c];
      REPEAT FINISHED => input.length ← 0;
      ENDLOOP;
    IF input.length # 0 THEN
      BEGIN
      j ← 0;
      WHILE (i←i+1) < input.length DO
        input[j] ← input[i]; j ← j+1;
        ENDLOOP;
      input.length ← j;
      END;
    FOR i IN [0..switches.length) DO -- convert all to lower case
      IF (c←switches[i]) IN ['A..'Z] THEN switches[i] ← c + ('a-'A);
      ENDLOOP;
    RETURN[TRUE];
    END;


  ReadCmdStream: PROCEDURE [stream: StreamHandle, name, switches: STRING] RETURNS [BOOLEAN] =
    BEGIN
    s: STRING ← [80];
    c: CHARACTER;
    DO
      c ← stream.get[stream ! StreamDefs.StreamError => EXIT];
      SELECT c FROM
        IODefs.SP => IF s.length # 0 THEN EXIT;
        IODefs.CR => EXIT;
        ENDCASE => StringDefs.AppendChar[s,c];
      ENDLOOP;
    RETURN[ReadCmdString[s,name,switches]];
    END;

  RequestFiles: ImageDefs.CleanupProcedure =
    BEGIN OPEN SegmentDefs;
    IF why # Save THEN RETURN;
    comcmRequest.file ← tableRequest.file ← NIL;
```

```
    ImageDefs.AddFileRequest[@comcmRequest];
    ImageDefs.AddFileRequest[@tableRequest];
    END;

  LookForCommands: PROCEDURE =
    BEGIN OPEN StreamDefs;
    s: STRING ← [100];
    ImageDefs.RemoveCleanupProcedure[@Cleanup];
    IF comcmRequest.file = NIL THEN RETURN;
    SegmentDefs.LockFile[comcmRequest.file];
    comcm ← CreateByteStream[comcmRequest.file,Read];
    [] ← ReadCmdStream[comcm,s,s];
    IF comcm.endof[comcm] THEN
      BEGIN comcm.destroy[comcm]; comcm ← NIL END;
    END;

  RepeatCommand: PROCEDURE =
    BEGIN OPEN IODefs, data;
    IF comcm=NIL THEN RETURN;
    WriteString["
Binding "L]; WriteString[source];
    IF copycode THEN
      BEGIN
      WriteString[", code to "L];
      WriteString[IF codefile.length = 0 THEN "BCD"L ELSE codefile];
      END;
    IF copysymbols THEN
      BEGIN
      WriteString[", "L];
      IF data.compress THEN WriteString["compressed "];
      WriteString["symbols to "L];
      WriteString[IF symbolfile.length = 0 THEN "BCD"L ELSE symbolfile];
      END;
    WriteChar[CR];
    RETURN
    END;

  SetSymbolCompression: PROCEDURE =
    BEGIN OPEN SymbolCompressorDefs, ControlDefs;
    IF LOOPHOLE[CompressSymbols,ProcDesc].tag = unbound THEN
      BEGIN OPEN IODefs;
      WriteLine["Symbol compression not available in this version"L];
      SIGNAL Rubout;
      END;
    data.compress ← TRUE;
    END;

  GetCommand: PROCEDURE =
    BEGIN
    done: BOOLEAN ← FALSE;
    line: STRING ← [100];
    file: STRING ← [40];
    c: CHARACTER;
    sense: BOOLEAN;
    sw: STRING ← [10];
    GetItem: PROCEDURE RETURNS [BOOLEAN] =
      BEGIN
      RETURN[
        IF comcm=NIL THEN ReadCmdString[line,file,sw]
        ELSE ReadCmdStream[comcm,file,sw]]
      END;
    GetSwitch: PROCEDURE RETURNS [c: CHARACTER] =
      BEGIN
      sense ← TRUE;
      WHILE i < sw.length DO
        c←sw[i];
        i ← i + 1;
        IF c = '- OR c = '~ THEN sense ← ~sense
        ELSE EXIT;
        ENDLOOP;
      RETURN
      END;

    IF comcm = NIL THEN
      BEGIN OPEN IODefs;
      WriteString["
```

```
Bind: "];
      ReadLine[line]; WriteChar[CR];
      END;
    source.length ← 0;
    data.codefile.length ← data.symbolfile.length ← data.outputfile.length ← 0;
    data.compress ← data.copycode ← data.copysymbols ← data.debug ← FALSE;
    localPause ← FALSE;
    file.length ← 0;
    WHILE file.length = 0 DO
      IF ~GetItem[] THEN RETURN;
      i ← 0;
      WHILE i < sw.length DO
        SELECT GetSwitch[] FROM
          'd => IF file.length = 0 THEN MiscDefs.CallDebugger[NIL]
                ELSE data.debug ← sense;
          'c => data.copycode ← TRUE;
          's => data.copysymbols ← TRUE;
          'x => BEGIN data.copysymbols ← TRUE; SetSymbolCompression[] END;
          'g => done ← TRUE;
          'p => IF file.length = 0 THEN globalPause ← sense
                ELSE localPause ← sense;
          'r =>
            BEGIN
            j: CARDINAL;
            FOR j IN [i..sw.length) DO sw[j-i] ← sw[j]; ENDLOOP;
            sw.length ← sw.length - i;
            DefaultFileName[file, ".image"L];
            TestPause[];
            Run[file, sw ! UNWIND => NULL; ANY => GO TO barf];
            END;
          ENDCASE => GO TO barf;
        REPEAT barf => SIGNAL IODefs.Rubout;
        ENDLOOP;
      ENDLOOP;
    DefaultFileName[file,".config"L];
    StringDefs.AppendString[source,file];
    WHILE ~done AND GetItem[] DO
      i ← 0;
      WHILE i < sw.length DO
        SELECT (c←GetSwitch[]) FROM
          'c =>
            BEGIN
            DefaultFileName[file,".code"L];
            data.codefile.length ← 0;
            StringDefs.AppendString[data.codefile,file];
            data.copycode ← TRUE;
            END;
          's,'x =>
            BEGIN
            DefaultFileName[file,".symbols"L];
            data.symbolfile.length ← 0;
            StringDefs.AppendString[data.symbolfile,file];
            data.copysymbols ← TRUE;
            IF c='x THEN SetSymbolCompression[];
            END;
          'o =>
            BEGIN
            DefaultFileName[file,".bcd"L];
            data.outputfile.length ← 0;
            StringDefs.AppendString[data.outputfile,file];
            END;
          'g => done ← TRUE;
          ENDCASE => SIGNAL IODefs.Rubout;
        ENDLOOP;
      ENDLOOP;
    END;

  Run: PROCEDURE [name, otherswitches: STRING] =
    BEGIN OPEN SegmentDefs;
    c: CHARACTER;
    i: CARDINAL;
    copy: StreamDefs.StreamHandle;
    copy ← StreamDefs.CreateByteStream[comcmRequest.file, Write+Append];
    FOR i IN [0..name.length) DO copy.put[copy, name[i]] ENDLOOP;
    IF otherswitches.length # 0 THEN
      BEGIN
```

```
      copy.put[copy, '/];
      FOR i IN [0..otherswitches.length) DO
        copy.put[copy, otherswitches[i]];
        ENDLOOP;
      END;
    copy.put[copy, ' ];
    IF comcm = NIL THEN copy.put[copy, IODefs.CR]
    ELSE DO
      c ← comcm.get[comcm ! StreamDefs.StreamError => GO TO done];
      copy.put[copy, c];
      REPEAT done => comcm.destroy[comcm];
      ENDLOOP;
    copy.destroy[copy];
    IF ExtensionIs[name, ".run"L] THEN
      BEGIN
      p: POINTER ← OsStaticDefs.OsStatics.EventVector;
      evi: TYPE = MACHINE DEPENDENT RECORD [
        type: [0..7777B], length: [0..17B]];
      p↑ ← evi[6,StringDefs.WordsForBcplString[name.length]+1];
      StringDefs.MesaToBcplString[name,p+1];
      ImageDefs.StopMesa[];
      END
    ELSE ImageDefs.RunImage[
      NewFileSegment[NewFile[name, Read, DefaultVersion], 1, 1, Read]];
    END;

  ExtensionIs: PROCEDURE [name, ext: STRING] RETURNS [BOOLEAN] =
    BEGIN
    t: STRING ← [40];
    i: CARDINAL;
    IF name.length <= ext.length THEN RETURN[FALSE];
    FOR i IN[name.length-ext.length..name.length) DO
      StringDefs.AppendChar[t,name[i]];
      ENDLOOP;
    RETURN[StringDefs.EquivalentString[t,ext]]
    END;

  TestPause: PROCEDURE =
    BEGIN OPEN IODefs;
    IF mustwait AND globalPause THEN
      BEGIN
      WriteString["Type any character to exit"L];
      [] ← ReadChar[];
      END;
    END;

  WriteHerald: PROCEDURE =
    BEGIN OPEN TimeDefs, IODefs;
    t: STRING ← [20];
    WriteString["Alto/Mesa Binder 4.1 of "L];
    AppendDayTime[t,UnpackDT[data.binderVersion.time]];
    t.length ← t.length - 3;
    WriteString[t];
    WriteChar[CR];
    t.length ← 0;
    AppendDayTime[t,UnpackDT[[0,0]]];
    t.length ← t.length - 3;
    WriteString[t];
    END;

  UncaughtSignal: PROCEDURE =
    BEGIN OPEN IODefs;
    message, signal: UNSPECIFIED;
    BcdErrorDefs.Error[error,"Fatal Binder Error"L];
    [message,signal] ← SIGNAL TrapDefs.SendMsgSignal;
    WriteString["  signal = "L];
    WriteOctal[signal];
    WriteString[", message = "L];
    WriteDecimal[message];
    WriteString[" ("L];
    WriteOctal[message];
    WriteChar[')];
    WriteChar[CR];
    END;
```

```
  i: CARDINAL;
  parsed, aborted: BOOLEAN;
  source: STRING ← [40];
  root: BcdTreeDefs.TreeLink;
  mustwait: BOOLEAN ← FALSE;

  -- request files
  ImageDefs.AddCleanupProcedure[@Cleanup];
  STOP; -- Wait for file lookup

  -- delete binder builder
  RemoveCode[ControlDefs.GetReturnLink[]];

  LookForCommands[];
  InitTableSegment[];

  data.network ← MiscDefs.GetNetworkNumber[];
  data.host ← OsStaticDefs.OsStatics.SerialNumber;

  WriteHerald[];

  DO BEGIN OPEN IODefs;
    GetCommand[!Rubout => GOTO Abort];
    data.starttime ← BcdUtilDefs.TimeNow[];
    IF source.length = 0 THEN EXIT;
    data.rootfile.length ← 0;
    FOR i IN [0..source.length) DO
      IF source[i] = '. THEN EXIT;
      StringDefs.AppendChar[data.rootfile, source[i]];
      ENDLOOP;
    IF data.outputfile.length = 0 THEN
      BEGIN
      StringDefs.AppendString[data.outputfile, data.rootfile];
      StringDefs.AppendString[data.outputfile, ".bcd"];
      END;
    data.sourcestream ← NewByteStream[source,Read ! ANY => GO TO NoFile];
    data.textdisplay ← FALSE;
    Initialize[];
      BEGIN ENABLE
        BEGIN
        BcdErrorDefs.GetModule => RESUME[BcdDefs.MTNull];
        BcdErrorDefs.GetInterface => RESUME[BcdDefs.EXPNull];
        BcdErrorDefs.GetSti => RESUME[BcdTabDefs.STNull];
        TableDefs.TableOverflow =>
          IF tablePages < TablePageLimit THEN
            BEGIN
            LoadTable[tablePages+TablePageStep];
            RESUME[[origin: tableOrigin, size: tableSize]];
            END
          ELSE
            BEGIN data.errors ← TRUE; IF ~data.debug THEN GOTO overflow END;
        UNWIND => Finalize[];
        ANY => IF ~data.debug THEN BEGIN UncaughtSignal[]; GOTO error END
        END;
      BEGIN OPEN SegmentDefs;
      SwapIn[parseg];
      [complete:parsed, errors: data.errors] ←
        BcdLALRDefs.Parse[FileSegmentAddress[parseg]];
      RemoveCode[BcdLALRDefs.Parse];
      Unlock[parseg]; SwapOut[parseg];
      END;
      IF ~parsed THEN GO TO Failed;
      root ← BcdTreeDefs.mlpop[]; RemoveCode[BcdTreeDefs.mlpop];
      BcdControlDefs.BuildSemanticEntries[root];
        RemoveCode[BcdControlDefs.BuildSemanticEntries];
      data.outputfti ← BcdUtilDefs.EnterFile[data.outputfile];
      BcdFileDefs.BuildFileTable[];
      BcdControlDefs.LoadRoot[root];
      BcdControlDefs.BindRoot[];
        RemoveCode[BcdControlDefs.BindRoot];
        RemoveCode[BcdControlDefs.LoadRoot];
      IF data.errors THEN WriteLine["Errors detected; BCD not written"]
      ELSE
        BEGIN
        BcdControlDefs.WriteBcd[root];
        IF data.errors THEN WriteLine["Errors detected; BCD file is invalid"];
```

```
      END;
    RemoveCode[BcdControlDefs.WriteBcd];
    BcdFileDefs.EraseFileTable[];
    EXITS
      Failed => data.errors ← aborted ← TRUE;
    END;
  Finalize[];
  IF aborted THEN WriteLine[" Failed!"];
  EXITS
    error => NULL;
    overflow => IODefs.WriteLine["Storage Overflow!"];
    NoFile =>
      BEGIN OPEN IODefs;
      WriteChar[CR];
      WriteString["Can't open "];
      WriteLine[source];
      data.errors ← TRUE;
      END;
    Abort => BEGIN data.errors ← TRUE; IODefs.WriteChar['?] END;
  END;
  IF comcm # NIL AND (data.errors OR data.warnings) THEN
    BEGIN
    mustwait ← TRUE;
    IF localPause THEN
      BEGIN OPEN IODefs;
      WriteString["Type any character to proceed"];
      [] ← ReadChar[];
      END;
    END;
  ENDLOOP;

TestPause[];
ImageDefs.StopMesa[];

END.
```